Chapter- 1

INTRODUCTION

A lot has been said during the past several years about how precision medicine and, more concretely, how genetic testing is going to disrupt the way diseases like cancer are treated.

But this is only partially happening due to the huge amount of manual work still required. Memorial Sloan Kettering Cancer Center (MSKCC) launched this competition, accepted by the NIPS 2017 Competition Track .

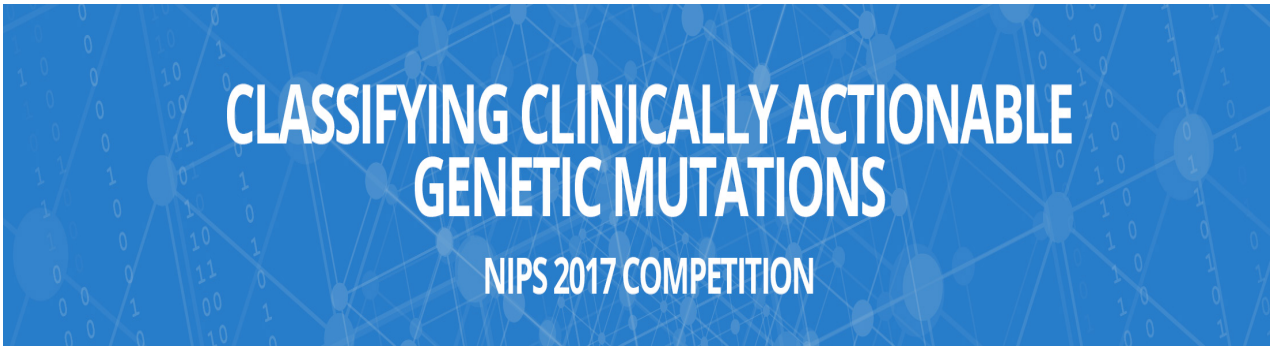


Figure 1: MSKCC PROBLEM

Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers).

Currently this interpretation of genetic mutations is being done manually. This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

For this competition MSKCC is making available an expert-annotated knowledge base where world-class researchers and oncologists have manually annotated thousands of mutations.

We need your help to develop a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations.

Chapter- 2

WORKFLOW

- A molecular pathologist selects a list of genetic variations of interest that he/she want to analyze.
- The molecular pathologist searches for evidence in the medical literature that somehow are relevant to the genetic variations of interest.
- Finally this molecular pathologist spends a huge amount of time analyzing the evidence related to each of the variations to classify them.

Our goal here is to replace step 3 by a machine learning model. The molecular pathologist will still have to decide which variations are of interest, and also collect the relevant evidence for them. But the last step which is also the most time consuming, will be fully automated.

2.1 REAL WORLD OBJECTIVES AND CONSTRAINTS:

1. It's a task of classification of each data points into one of nine cancer classes correctly.
2. With the prediction of label we also need probability of each data point belonging to any of the nine classes.
3. Errors can be costly.
4. Interpretability is also important.
5. Make the log-Loss < 1.0.

A gene can have thousands of different point mutations. Some of them matter for a specific type of cancer, e.g. EGFR for Breast Cancer, and some others don't matter at all.
A point mutation just represent a nucleotide change, there is an adenine (A) where there should be a cytosine (C) for instance. And these can happen in multiple genes as well.

But a pair of Gene and Variation, define a set of specific genomic coordinates. For instance, PIK3CA, H1047R, means that if you pick the gene PIK3CA and start counting from the beginning. The codon (triplets of nucleotides) number 1047, Histidine is changed to a Arginine.

Chapter- 3

MACHINE LEARNING PROBLEM FORMULATION

In this project we will develop algorithms to classify genetic mutations based on clinical evidence (text).

There are nine different classes a genetic mutation can be classified on.

This is not a trivial task since interpreting clinical evidence is very challenging even for human specialists. Therefore, modeling the clinical evidence (text) will be critical for the success of your approach.

Both, training and test, data sets are provided via two different files. One (training/test_variants) provides the information about the genetic mutations, whereas the other (training/test_text) provides the clinical evidence (text) that our human experts used to classify the genetic mutations. Both are linked via the ID field.

Therefore the genetic mutation (row) with ID=15 in the file training_variants, was classified using the clinical evidence (text) from the row with ID=15 in the file training_text

Finally, to make it more exciting!! Some of the test data is machine-generated to prevent hand labeling. You will submit all the results of your classification algorithm, and we will ignore the machine-generated samples.

3.1 Data files information:

training_variants - 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 amino acid change for this mutations), Class (1-9 the class this genetic mutation has been classified on). training_variants (**ID, GENE, VARIATIONS, CLASS**).

training_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation). training_text(**ID, Text**).

Both these data files are have a common column called **ID.**

Example:-

TRAINING_ VARIANTS :	**ID, Gene, Variation, Class**
	0, FAMA58A, Truncating Mutations, 1

1, CBL, W802*, 2
2, CBL, Q249E, 2
3, XYXXY, QWEW, 7

TRAINING_ TEXT : **ID, TEXT**

0|| report here a novel update (2012–2016) of the genetic screening of the large AD-EOAD series ascertained across 28 French hospitals from 1993 onwards, bringing the total number of families with identified mutations to *n* = 170. Families were included when at least two first-degree relatives suffered from early-onset Alzheimer disease (EOAD) with an age of onset (AOO) ≤65 y in two generations. Furthermore, we also screened 129 sporadic cases of Alzheimer disease with an AOO below age 51 (44% males, mean AOO = 45 ± 2 y). *APP*, *PSEN1*, or *PSEN2* mutations were identified in 53 novel AD-EOAD families. Of the 129 sporadic cases screened, 17 carried a *PSEN1* mutation and 1 carried an *APP* duplication (13%). Parental DNA was available for 10 sporadic mutation carriers, allowing us to show that the mutation had occurred de novo in each case. Thirteen mutations (12 in *PSEN1* and 1 in *PSEN2*) identified either in familial or in sporadic cases were previously unreported. Of the 53 mutation carriers with available cerebrospinal fluid (CSF) biomarkers, 46 (87%) had all three CSF biomarkers—total tau protein (Tau), phospho-tau protein (P-Tau), and amyloid β $(Aβ)_{42}$—in abnormal ranges. No mutation carrier had the three biomarkers in normal ranges. One limitation of this study is the absence of functional assessment of the possibly and probably pathogenic variants, which should help their classification

- There are 9 different classes a genetic mutation can be classified into=>
- Multiclass classification problem. (1, 2, 3, 4……9).
- Suppose 3 and 7 associated for cancer.

3.2 Probability Table:

Class	1	2	3	4	5	6	7	8	9
probability	P1	P2	P3	P4	P5	P6	P7	P8	P9

3.3 Mapping the real world problem to an ML problem:

1. **Type of Probem:** There are 9 different Classes a Genetic Mutation can be classified into i.e Multiclass Classification Problem.

2. **Performance Metrices**

   I. Multiclass Logloss
   II. Confusion matrix

3. Machine learning objectives and constraints

Objective:

Predict the probability of each data-point belonging to each of the 9 classes.

Constraints:

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

4. Train, CV and Test data set : Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively and do the same operation for all the three and analyze the result.

Chapter- 4

PERFOMANCE METRICES

4.1 Multi class log-loss

Logarithmic Loss, or simply Log Loss, is a classification_log_function often used as an evaluation metric in kaggle competitions. Since success in these competitions hinges on effectively minimizing the Log Loss, it makes sense to have some understanding of how this metric is calculated and how it should be interpreted.

Log Loss quantifies the accuracy of a classifier by penalising false classifications. Minimising the Log Loss is basically equivalent to maximizing the accuracy of the classifier, but there is a subtle twist which we'll get to in a moment.

In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class. Mathematically Log Loss is defined as:-

$$\frac{-1}{N}\sum_{I=1}^{N}\sum_{J=1}^{M} \mathrm{Y_{IJ}} \log \mathrm{p_{ij}}$$

where N is the number of samples or instances, M is the number of possible labels, Y_{ij} log loss is a binary indicator of whether or not label j is the correct classification for instance i, and p_{ij} is the model probability of assigning label j to instance i. A perfect classifier would have a Log Loss of precisely zero. Less ideal classifiers have progressively larger values of Log Loss. If there are only two classes then the expression above simplifies to

$$\frac{-1}{N}\sum_{I=1}^{N}[y_i log\ p_i + (1-y_i)\log\ (1-p_i)]$$

Note that for each instance only the term for the correct class actually contributes to the sum.

4.2 Confusion matrix:

A confusion matrix is a table that is often used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

I wanted to create a **"quick reference guide" for confusion matrix terminology** because I couldn't find an existing resource that suited my requirements: compact in presentation, using numbers instead of arbitrary variables, and explained both in terms of formulas and sentences.

Let's start with an **example confusion matrix for a binary classifier** (though it can easily be extended to the case of more than two classes):

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Figure 2: Confusion Matrix

What can we learn from this matrix?

- There are two possible predicted classes: "yes" and "no". If we were predicting the presence of a disease, for example, "yes" would mean they have the disease, and "no" would mean they don't have the disease.
- The classifier made a total of 165 predictions (e.g., 165 patients were being tested for the presence of that disease).
- Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.
- In reality, 105 patients in the sample have the disease, and 60 patients do not.

Let's now define the most basic terms, which are whole numbers (not rates):

- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they don't have the disease.
- **false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

Chapter-5

Exploratory Data Analysis

5.1 Reading Data: We first need to read the data files we have. We have two data set training_variants and training text.

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

```
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()
```

Output

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

Out[2]:

	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

Figure 3: Training variant Data

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 amino acid change for this mutations
Class : 1-9 the class this genetic mutation has been classified on

II. Reading training_text:

```
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()
```

OUT [3]:

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

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

Figure 4: Training text Data

5.2 Preprocessing of text: After reading the data we need to do certain operations the training_text data as follows:-

- Removing **stop words** from the text. Stop words are am, is, are, the, has, have… etc. we don't need stop words in the text data, so after removing all these words we will have to process lesser data and this results optimization in execution.
- Replace all **special characters** with space.
- replace multiple spaces with single space.
- converting all the chars into lower-case.

```
# 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
#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")
```

-merging both gene_variations and text data based on ID

```
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

OUT [5]:

	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...

Figure 5: Merging Training variants and Training text

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

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

In [16]:

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

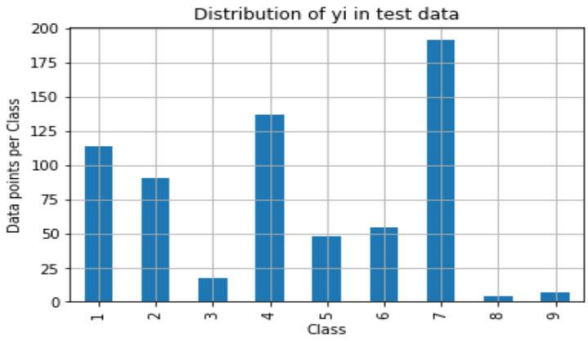
```
# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

In [17]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()
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.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(',
np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()
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], '(',
np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')
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()
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]/cv_df.shape[0]*100), 3), '%)')
```

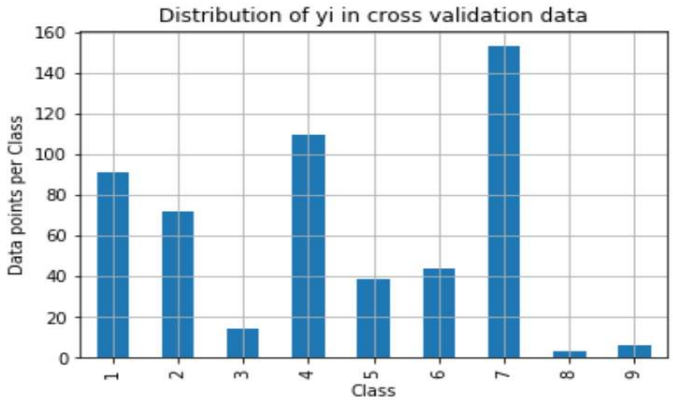
OUT [17]




```
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 %)
---------------------------------------------------------------------------
```

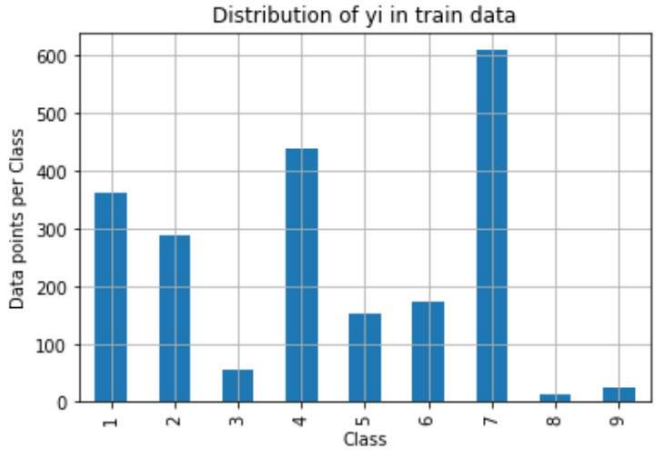
:

Figure 6: Class Distribution in test data




```
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 %)
```

Figure 7: Class distribution in cross validation data




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

Figure 8: Class distribution in Train data

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

Log loss on Cross Validation Data using Random Model : 2.493666690798053.
Log loss on Test Data using Random Model: 2.445183264544613.

Log loss of random model is approximately 2.5, so we need the machine learning model with log loss less than 2.5.

5.3 Univariate analysis:

we need to analyze all features of our data set training_variants and training text.

-Univariate Analysis on Gene Feature

Q. Gene, What type of feature it is ?
Ans. Gene is a categorical variable.
Q. How many categories are there?
Ans.There are 232 different categories of genes in the train data.
Q. How to featurize this Gene feature ?
Ans.there are two ways we can featurize this variable:

1. One hot Encoding
2. Response coding

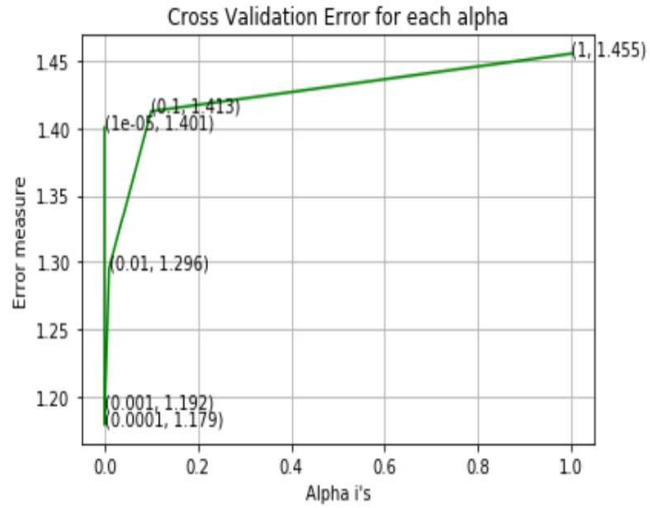
We will choose the One hot Encoding featurization based here for now. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.
In [21]:

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

# know  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.

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

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

OUT [21]:

```
For values of alpha =  1e-05 The log loss is: 1.4005555527470852
For values of alpha =  0.0001 The log loss is: 1.179340161820206
For values of alpha =  0.001 The log loss is: 1.1920606034079948
For values of alpha =  0.01 The log loss is: 1.2955107368017145
For values of alpha =  0.1 The log loss is: 1.412544823106519
For values of alpha =  1 The log loss is: 1.4552582366725741
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0415595201966887
For values of best alpha =  0.0001 The cross validation log loss is: 1.179340161820206
For values of best alpha =  0.0001 The test log loss is: 1.2086527043791122
```

Figure 9: Cross Validation error for each alpha in Gene feature

-Univariate Analysis on Variation Feature

Q. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q. How many categories are there?

There are 1916 different categories of variations in the train data.

Q.How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable.

1. One hot Encoding

2. Response coding

We will be using One hot Encoding method to featurize the Variation Feature

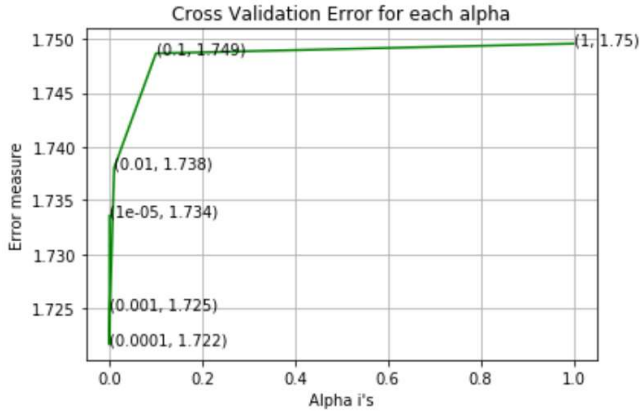
In [22]:

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

# 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.

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

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

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

OUT [22]:

```
For values of alpha =  1e-05 The log loss is: 1.7335721680090286
For values of alpha =  0.0001 The log loss is: 1.72166551571151
For values of alpha =  0.001 The log loss is: 1.7249727158308956
For values of alpha =  0.01 The log loss is: 1.7380948794689282
For values of alpha =  0.1 The log loss is: 1.748669375869514
For values of alpha =  1 The log loss is: 1.7495674317353667
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7456583822237609
For values of best alpha =  0.0001 The cross validation log loss is: 1.72166551571151
For values of best alpha =  0.0001 The test log loss is: 1.7242034752511008
```

Figure 10: Cross validation error for each alpha in variation feature

-Univariate analysis on number of words feature

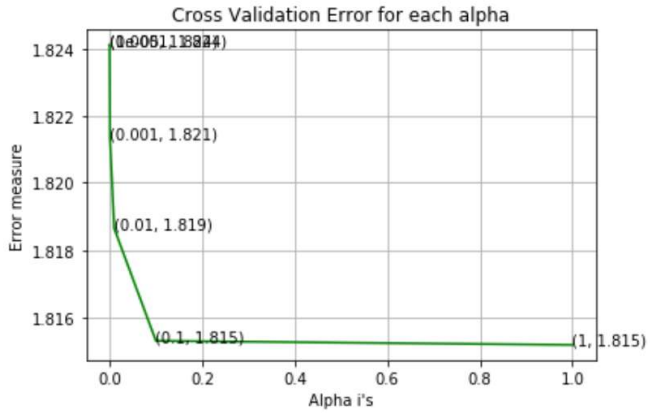
IN [23]:

```
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_df["Feature_1"].reshape(-1,1), y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Feature_1.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Feature_1.reshape(-1,1))
    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_df.Feature_1.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Feature_1.reshape(-1,1), y_train)
predict_y = sig_clf.predict_proba(train_df.Feature_1.reshape(-1,1))
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_df.Feature_1.reshape(-1,1))
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_df.Feature_1.reshape(-1,1))
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))
```

OUT [23]:

```
For values of alpha =  1e-05 The log loss is: 1.8241096612789325
For values of alpha =  0.0001 The log loss is: 1.8241096612794818
For values of alpha =  0.001 The log loss is: 1.8213303970404182
For values of alpha =  0.01 The log loss is: 1.8186476370872977
For values of alpha =  0.1 The log loss is: 1.8153239532039447
For values of alpha =  1 The log loss is: 1.8152017039678854
```




```
For values of best alpha =  1 The train log loss is: 1.8145762602898243
For values of best alpha =  1 The cross validation log loss is: 1.8152017039678854
For values of best alpha =  1 The test log loss is: 1.8027729211835348
```

Figure 11: Cross Validation for each alpha in number of words feature

-Univariate analysis on Number of character

-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 classes?
5. Is the text feature stable across train, test and CV datasets?

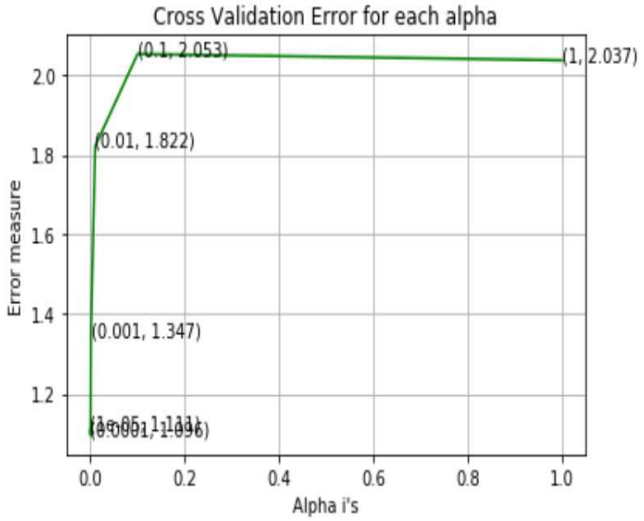
In [25]:

```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]
# default parameters
rt=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.
cv_log_error_array=[]
for i in alpha:
   clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
   clf.fit(train_text_feature_onehotCoding, y_train)
   sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
   sig_clf.fit(train_text_feature_onehotCoding, y_train)
   predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
   cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
   print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=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_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

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

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

OUT [25]:

```
For values of alpha =  1e-05 The log loss is: 1.1105363411250055
For values of alpha =  0.0001 The log loss is: 1.0964906815887412
For values of alpha =  0.001 The log loss is: 1.346817068493882
For values of alpha =  0.01 The log loss is: 1.8215906927561887
For values of alpha =  0.1 The log loss is: 2.0530451970564907
For values of alpha =  1 The log loss is: 2.0373670509665978
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7467683268790596
For values of best alpha =  0.0001 The cross validation log loss is: 1.0964906815887412
For values of best alpha =  0.0001 The test log loss is: 1.1013255697297029
```

Figure 12: Cross Validation for each alpha in Text feature

-One Hot Encoding:- One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. It is used for High dimensional data. It is used in Logistic Regression.

-Response Coding:- **Response Coding** is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. It is used for low dimensional data. It is used in random forest classifier.

Chapter 6

MACHINE LEARNING MODEL

Various machine learning models have been used in this project. Every model has certain errors (**log_loss** and **percentage misclassified**) in the prediction of classes. We are well familiar with log loss.

Percentage misclassified: chance in percentage that the class predicted by the model is not correct.

Thus we need to find the model which has log_loss is nearest to 0 and **Percentage misclassified** is minimum.

First we find best parameter for the model called **Hyper Parameter Tunning** (for example k in KNN), then we test the model using this parameter using one hot encoding.

6.1 Naive Bayes Classifier:

Principle of Naive Bayes Classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems etc. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent. In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

6.1.1 Hyper parameter tuning

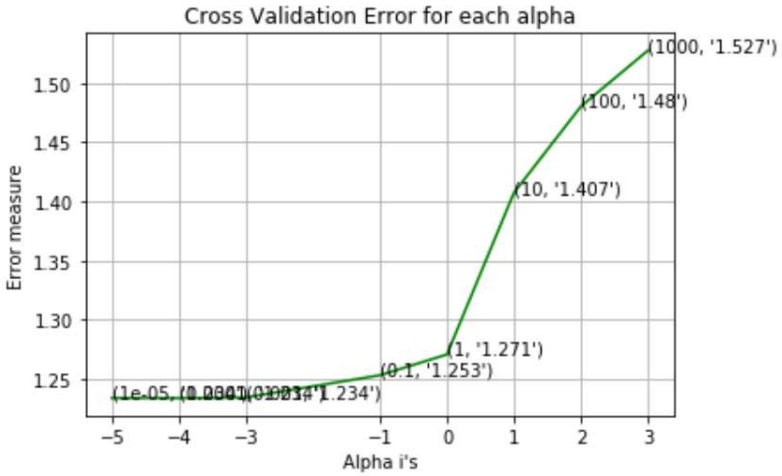
In [27]:

```
# 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.
# 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
# ----------------------------
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
   print("for alpha =", i)
   clf = MultinomialNB(alpha=i)
   clf.fit(train_x_onehotCoding, train_y)
   sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
   sig_clf.fit(train_x_onehotCoding, train_y)
   sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
   cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
   ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
```

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

OUT [28]:

```
for alpha = 1e-05
Log Loss : 1.233816172023287
for alpha = 0.0001
Log Loss : 1.2337083709494239
for alpha = 0.001
Log Loss : 1.2343450081698089
for alpha = 0.1
Log Loss : 1.253028536291359
for alpha = 1
Log Loss : 1.2705641127871883
for alpha = 10
Log Loss : 1.4071878034064196
for alpha = 100
Log Loss : 1.4801906371270999
for alpha = 1000
Log Loss : 1.5274644180242751
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6218903432935438
For values of best alpha =  0.0001 The cross validation log loss is: 1.2337083709494239
For values of best alpha =  0.0001 The test log loss is: 1.2642411605579797
```

Figure 13: Cross validation for each alpha in Naive Bayes Model

6.1.2. Testing the model with best hyper parameters

In [29]:

```
# 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.
# 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

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 estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.as_matrix()))
```

OUT [29]:

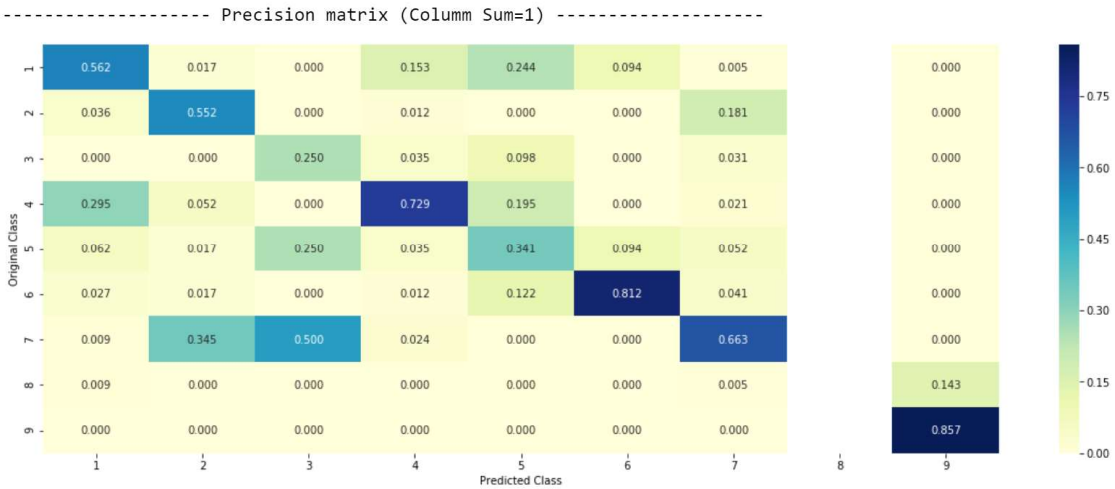


Figure 14: Precision Matrix for Naive Bayes Model

Predicted Class : 5
Predicted Class Probabilities: [[0.0899 0.0671 0.0139 0.1098 0.4708 0.0446 0.1952 0.0043 0.0045]]

Actual Class : 1

6.2 K nearest neighbor classifier:

In pattern recognition, the ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

k-NN is a type of instance-base learning , or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all machine learning algorithms

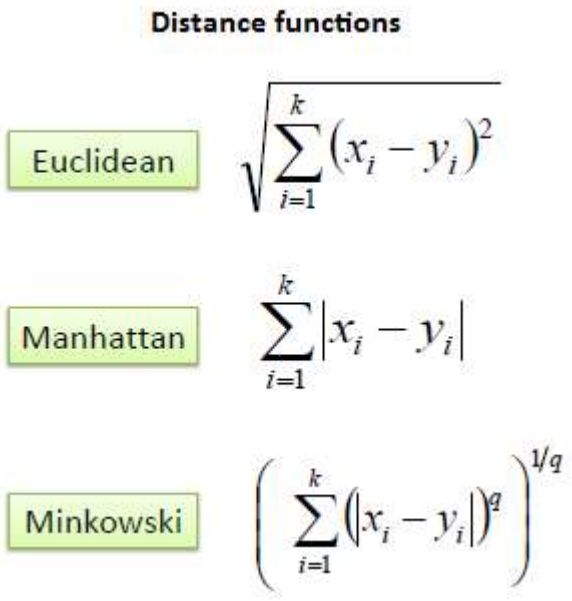


Figure 15: Distance functions in KNN

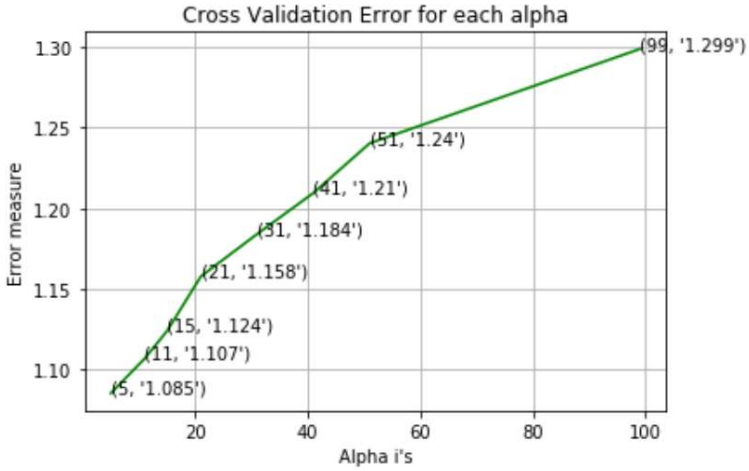
6.2.1. Hyper parameter tuning

In [30]:

```
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-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)

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_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

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

In [30]:




```
For values of best alpha =  5 The train log loss is: 0.8589002413672985
For values of best alpha =  5 The cross validation log loss is: 1.0852829601316618
For values of best alpha =  5 The test log loss is: 1.0951620282817154
```

Figure 16: Cross validation error in KNN model

For values of best alpha = 5 The train log loss is: 0.8589002413672985
For values of best alpha = 5 The cross validation log loss is: 1.0852829601316618
For values of best alpha = 5 The test log loss is: 1.0951620282817154

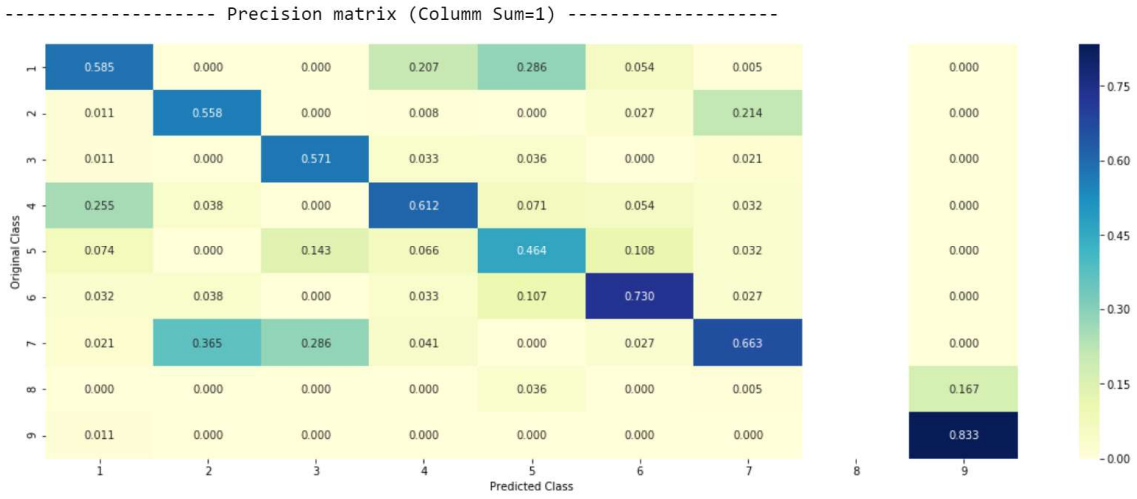
6.2.2. Testing the model with best hyper parameters

In [31]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
# methods of
```

```
# fit(X, y) : Fit the model using X as training data and y as target values
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

OUT [31]



:

Figure 17: Precision Matrix for KNN model

6.3 Logistic regression :-

it is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

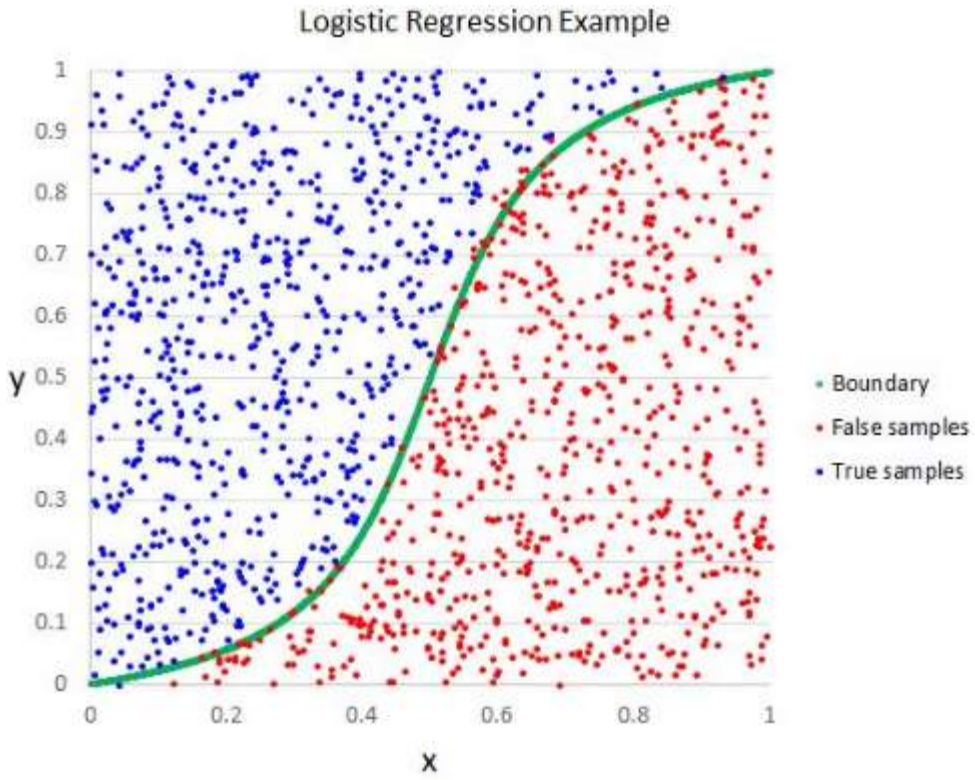


Figure 18: Logistic regression example

6.3.1 Hyper parameter tuning

In [32]:

```
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None,
tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
# some of methods
# fit(X, y[, coef_init, intercept_init, ...])            Fit linear model with Stochastic Gradient Descent.
# predict(X)          Predict class labels for samples in X.
#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-
intuition-1/
#-------------------------------
# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
```

```
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep])         Get parameters for this estimator.
# predict(X)         Predict the target of new samples.
# predict_proba(X)           Posterior probabilities of classification

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
   print("for alpha =", i)
   clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
   clf.fit(train_x_onehotCoding, train_y)
   sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
   sig_clf.fit(train_x_onehotCoding, train_y)
   sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
   cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
   # to avoid rounding error while multiplying probabilites we use log-probability estimates
   print("Log Loss :",log_loss(cv_y, sig_clf_probs))
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
   ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y,
labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y,
labels=clf.classes_, eps=1e-15))
```

6.3.2. Testing the model with best hyper paramters

In [33]:

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

OUT [33]:

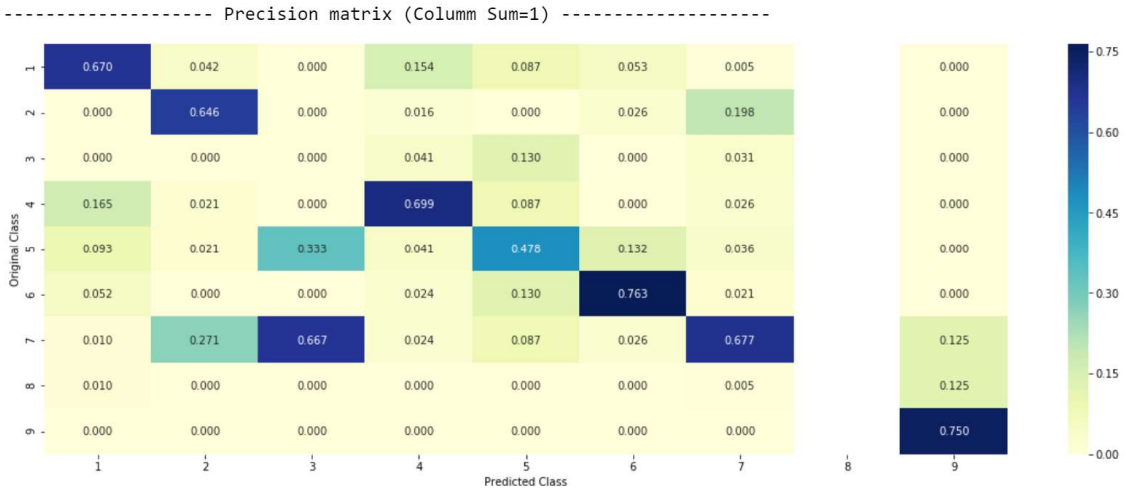


Figure 19: Precision Matrix for logistic regression

Predicted Class : 4 Predicted Class

Probabilities: [[0.0153 0.0083 0.1122 0.8122 0.0204 0.0098 0.0174 0.0019 0.0025]]

Actual Class : 4

6.4 Linear Support Vector machines:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

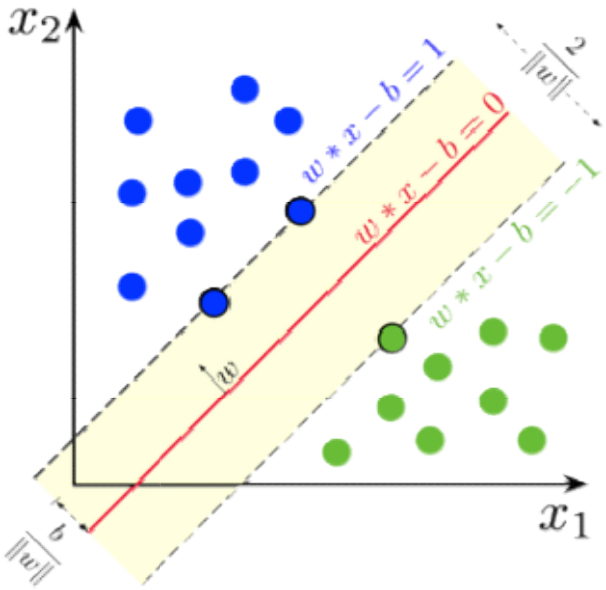


Figure 20: Linear SVM example

6.4.1. Hyper parameter tuning

In [36]:

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

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

# 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

alpha = [10 ** x for x in range(-5, 3)]
```

```
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
```

6.4.2. Testing model with best hyper parameters

In [37]:

```
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr',
random_state=None)
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

OUT [37]:

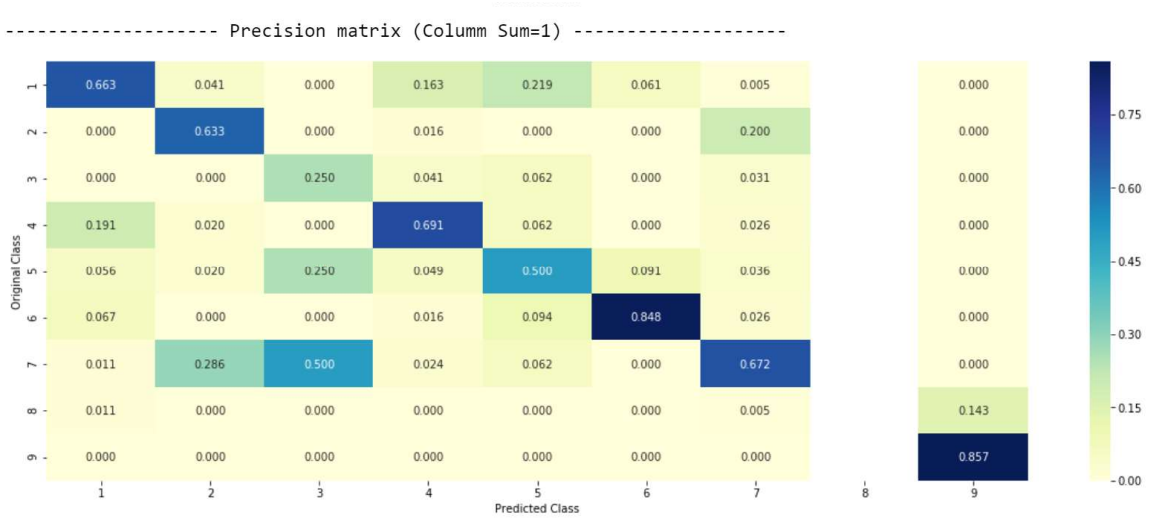


Figure 21: Precision matrix for Linear SVM model

Predicted Class : 4
Predicted Class Probabilities:
[[0.0313 0.016 0.0338 0.8273 0.0258 0.0116 0.0495 0.002 0.0027]]
Actual Class : 4

6.5 Random Forest classifier:

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it.

To say it in simple words: *Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.*

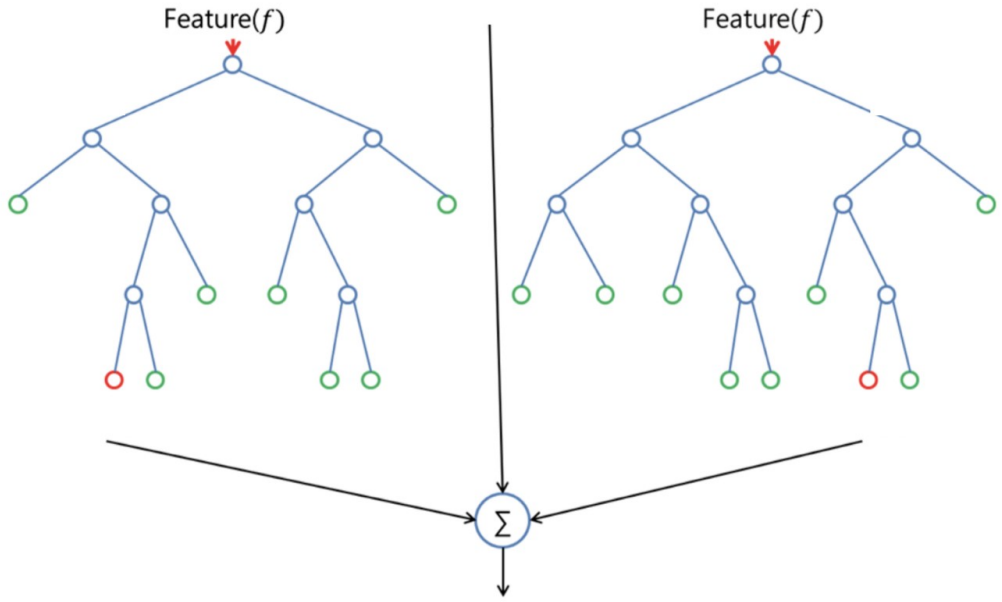


Figure 22: Random Forest Example

6.5.1. Hyper parameter tuning (With One hot Encoding)

In [39]:

```
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
warm_start=False,
# class_weight=None)
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X)          Perform classification on samples in X.
# predict_proba (X)           Perform classification on samples in X.
```

```
# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).
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
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
   for j in max_depth:
      print("for n_estimators =", i,"and max depth = ", j)
      clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42,
n_jobs=-1)
      clf.fit(train_x_onehotCoding, train_y)
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
      sig_clf.fit(train_x_onehotCoding, train_y)
      sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
      cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
      print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
   ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
```

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

In [40]:

```
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0,
# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

OUT [40]:

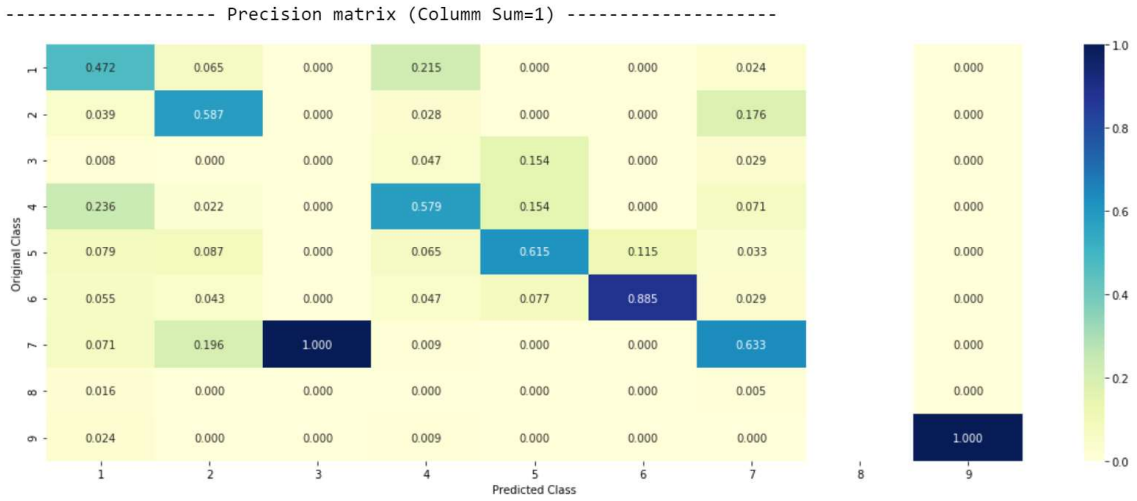


Figure 23: Precision matrix for Random Forest model

Predicted Class : 4
Predicted Class Probabilities:
[[0.0259 0.0089 0.0237 0.8624 0.0283 0.024 0.021 0.0024 0.0036]]
Actual Class : 4

Chapter-7

CONCLUSION

We tried many models for 2000 most important words, we were best able to classify Cancer into its actual classes with minimum error :-

- Stacking Model and Maximum Vote classifier model are hard to interpret.
- Best Model For this Dataset is :- **Logistic Regression with Class Balancing**
- Train Log-Loss:-0.4246
- CV Log-Loss:-0.9572
- Test Log-Loss:-0.9994
- Percentage Misclassified:-0.3270

Model	Train	CV	Test	Percentage Misclassified
Naive Bayes	0.6218	1.2337	1.2642	0.3759
KNN (Best K=11)	0.8589	1.0852	1.0951	0.3778
LR (With Class Balancing)	0.4246	0.9572	0.9994	0.3270
LR (Without Class Balancing)	0.5775	0.9554	0.9950	0.3402
LINEAR SVM	0.4841	1.0049	1.0301	0.3289
Random Forest	0.8653	1.1663	1.1736	0.4078

Figure 24: Conclusion Table

Chapter-8

REFERENCES

- ✓ https://www.kaggle.com/c/msk-redefining-cancer-treatment (Paper reference)
- ✓ Tutorial Point
- ✓ Wikipedia
- ✓ **Books:**
  - ❖ ***Learning Python*, Mark Lutz & David Ascher, O'Reilly Press, 1999**
  - ❖ ***Programming Python*, Mark Lutz, O'Reilly Press, 2006**
  - ❖ ***Core Python Programming (2nd Edition)*, Wesley J. Chun, Prentice Hall, 2006**
  - ❖ **Python for Data Analysis** by Wes McKinney
  - ❖ **Think Python** by Allen B. Downey
  - ❖ **Learn Python Programming the Easy and Fun Way** by ElaiyaIsweraLallan
- ✓ **http://docs.python.org - online version of built-in Python function documentation**
- ✓ **http://laurent.pointal.org/python/pqrc - Python Quick Reference Card**
- ✓ **http://rgruet.free.fr - long version of Python Quick Reference Card**
- ✓ **http://mail.python.org - extensive Python forum**
- ✓ **www.Google.com**
- ✓ **www.Kaggle.com**
- ✓ **https://en.wikipedia.org/wiki/Financial_crisis_of_2007%E2%80%9308.com**
- ✓ **https://www.python.org**
- ✓ **www.W3schools.com**
- ✓ **Learn Python (Programming Tutorial for Beginners) - Programiz**
- ✓ **www.codeacadmy.com**

➢ APPENDICES

1. Libraries used:

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
```

2. Prediction of classes using a 'Random' Model

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

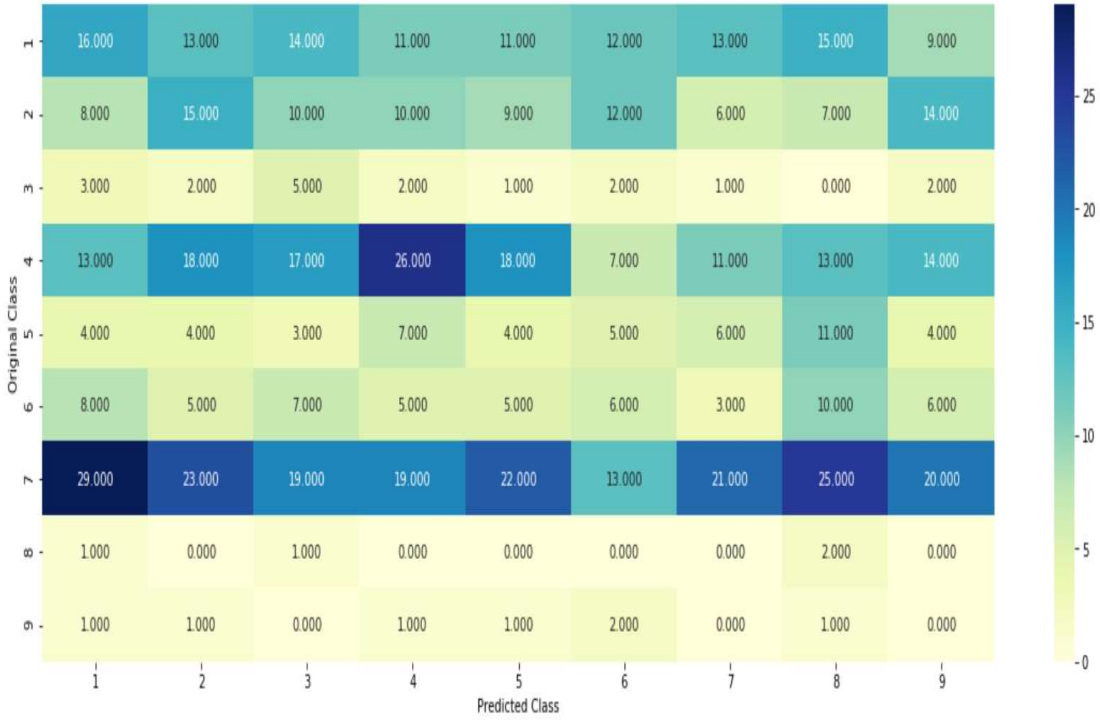
IN [18]:

```
# 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_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)
```

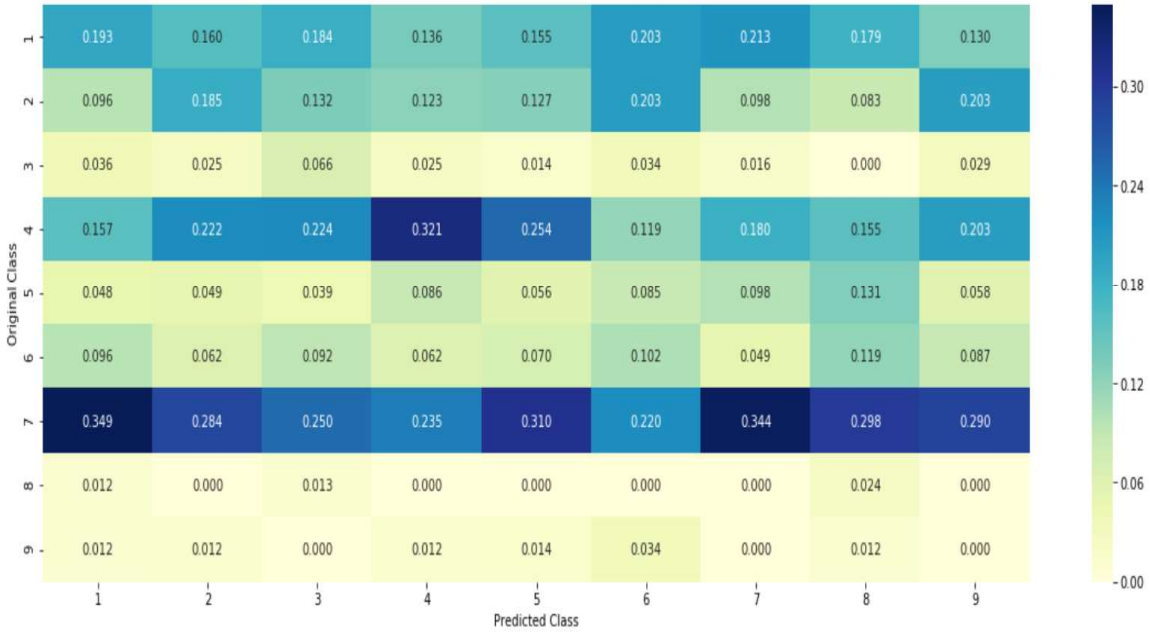
OUT [18]:

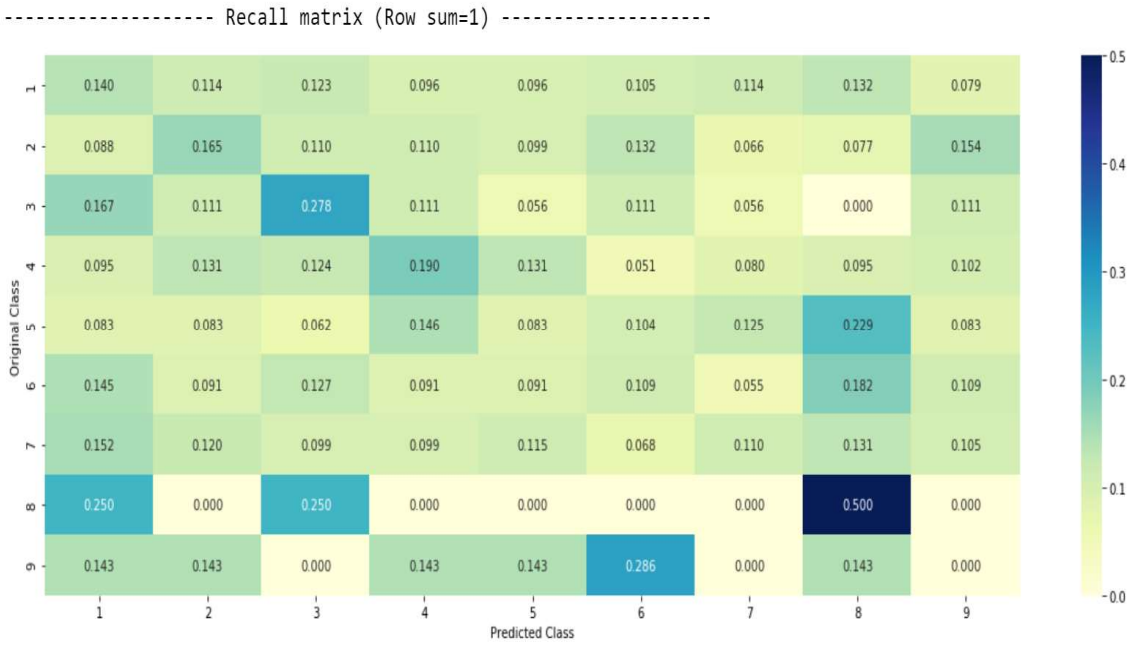
```
Log loss on Cross Validation Data using Random Model 2.493666690798053
Log loss on Test Data using Random Model 2.445183264544613
```

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



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





3. Univariate analysis on number of words feature

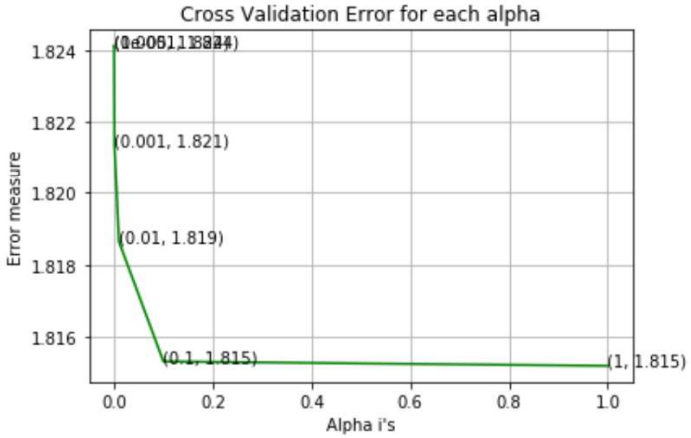
IN [23]:

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_df["Feature_1"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Feature_1.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Feature_1.reshape(-1,1))

    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_df.Feature_1.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Feature_1.reshape(-1,1), y_train)
predict_y = sig_clf.predict_proba(train_df.Feature_1.reshape(-1,1))
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_df.Feature_1.reshape(-1,1))
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_df.Feature_1.reshape(-1,1))
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))
```

OUT [23]:

```
For values of alpha =  1e-05 The log loss is: 1.8241096612789325
For values of alpha =  0.0001 The log loss is: 1.8241096612794818
For values of alpha =  0.001 The log loss is: 1.8213303970404182
For values of alpha =  0.01 The log loss is: 1.8186476370872977
For values of alpha =  0.1 The log loss is: 1.8153239532039447
For values of alpha =  1 The log loss is: 1.8152017039678854
```




```
For values of best alpha =  1 The train log loss is: 1.8145762602898243
For values of best alpha =  1 The cross validation log loss is: 1.8152017039678854
For values of best alpha =  1 The test log loss is: 1.8027729211835348
```

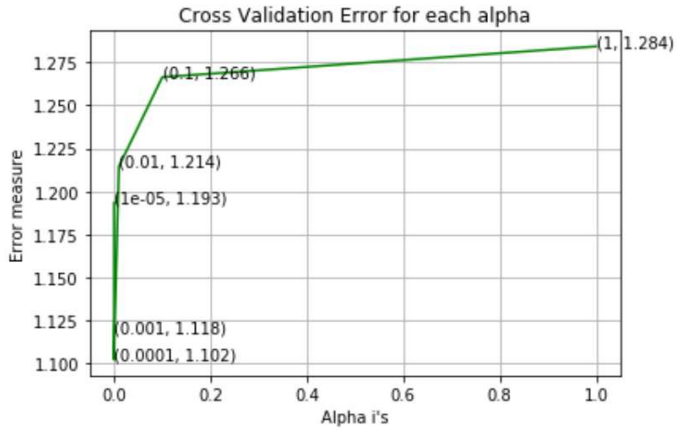
4. Univariate analysis on gene-variation feature

In [24]:

```
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(df_geneandvar_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_geneandvar_train, y_train)
    predict_y = sig_clf.predict_proba(df_geneandvar_cv)
    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(df_geneandvar_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(df_geneandvar_train, y_train)
predict_y = sig_clf.predict_proba(df_geneandvar_train)
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(df_geneandvar_cv)
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(df_geneandvar_test)
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))
```

OUT [24]:

```
For values of alpha =  1e-05 The log loss is: 1.1932901215272576
For values of alpha =  0.0001 The log loss is: 1.102290132342943
For values of alpha =  0.001 The log loss is: 1.1181306832005078
For values of alpha =  0.01 The log loss is: 1.2144516238587448
For values of alpha =  0.1 The log loss is: 1.2662628626166113
For values of alpha =  1 The log loss is: 1.2841147606722276
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5215962530649423
For values of best alpha =  0.0001 The cross validation log loss is: 1.102290132342943
For values of best alpha =  0.0001 The test log loss is: 1.1352915779532327
```

5.Machine Learning Models

In [26]:

 # this function will be used just for naive bayes

```
# 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(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")
```

6. Logistic Regression Without Class balancing

6.1 Hyper parameter tuning

In [34]:

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
```

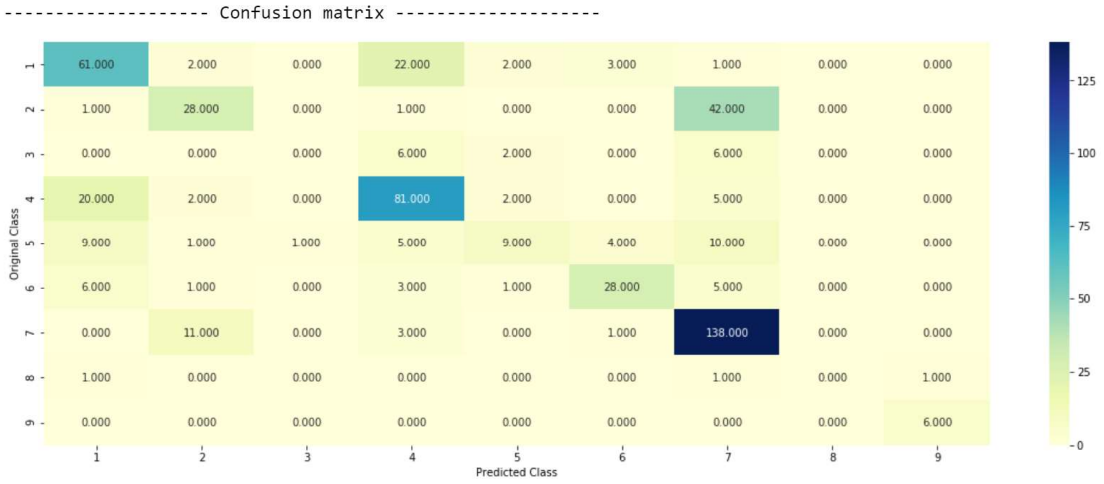
```
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y,
labels=clf.classes_, eps=1e-15))
```

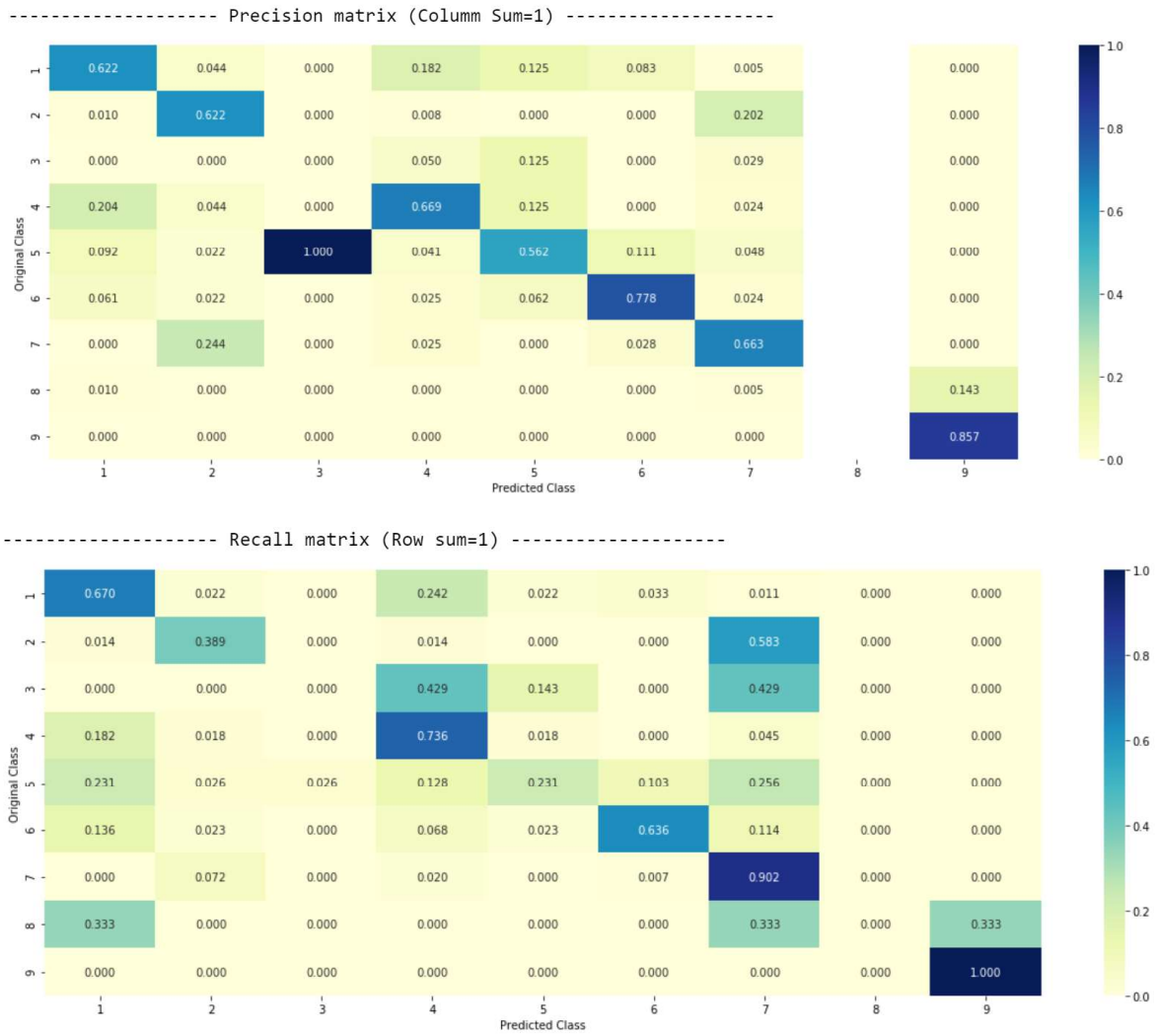
6.2 Testing model with best hyper parameters

In [35]:

```
# 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.

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

OUT [35]:





Predicted Class : 4
Predicted Class Probabilities:
[[0.0099 0.0071 0.0348 0.9187 0.0124 0.0065 0.0049 0.0023 0.0034]]
Actual Class : 4