PersonalizedCancerDiagnosis1

March 24, 2019

Personalized cancer diagnosis

1. Business Problem

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

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

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

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

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

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

2.1.2. Example Data Point
training_variants
ID,Gene,Variation,Class 0,FAM58A,Truncating Mutations,1 1,CBL,W802*,2 2,CBL,Q249E,2 ...
training_text

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

2.2. Mapping the real-world problem to an ML problem
2.2.1. Type of Machine Learning Problem

```
There are nine different classes a genetic mutation can be classified into => Multi clas
```

2.2.2. Performance Metric
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation
Metric(s): * Multi class log-loss * Confusion matrix
2.2.3. Machine Learing Objectives and Constraints
Objective: Predict the probability of each data-point belonging to each of the nine classes.
Constraints:

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

2.3. Train, CV and Test Datasets
Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

In [0]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from 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
```

In [ ]:
```
# Install the PyDrive wrapper & import libraries.
# This only needs to be done once per notebook.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Download a file based on its file ID.
#
# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
listed = drive.ListFile().GetList()
for file in listed:
    print('title {}, id {}'.format(file['title'], file['id']))
```

3.1. Reading Data

In [0]:
```
download = drive.CreateFile({'id': '1efUXMCeFbokuajSQWKXDchI47u4hqm9F'})
download.GetContentFile('result_prep.csv')
```

In [0]:
```
download = drive.CreateFile({'id': '1mMkUMKiNDvcgXRjuV9gbeoMmkZyeniTA'})
download.GetContentFile('training_text')
```

In [0]:
```
result=pd.read_csv('result_prep.csv')
```

3.1.1. Reading Gene and Variation Data

In [0]:
```
data = pd.read_csv('training/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[0]:    ID    Gene             Variation  Class
       0   0  FAM58A  Truncating Mutations      1
       1   1     CBL                 W802*      2
       2   2     CBL                 Q249E      2
       3   3     CBL                 N454D      3
       4   4     CBL                 L399V      4
```

```
training/training_variants is a comma separated file containing the description of the genetic m
Fields are
<ul>
    <li><b>ID : </b>the id of the row used to link the mutation to the clinical evidence</li>
    <li><b>Gene : </b>the gene where this genetic mutation is located </li>
    <li><b>Variation : </b>the aminoacid change for this mutations </li>
    <li><b>Class :</b> 1-9 the class this genetic mutation has been classified on</li>
</ul>
```

3.1.2. Reading Text Data

```
In [24]: # note the seprator in this file
         data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],s
         print('Number of data points : ', data_text.shape[0])
         print('Number of features : ', data_text.shape[1])
         print('Features : ', data_text.columns.values)
         data_text.head()
```

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

```
Out[24]:    ID                                               TEXT
         0   0  Cyclin-dependent kinases (CDKs) regulate a var...
         1   1   Abstract Background  Non-small cell lung canc...
         2   2   Abstract Background  Non-small cell lung canc...
         3   3  Recent evidence has demonstrated that acquired...
         4   4  Oncogenic mutations in the monomeric Casitas B...
```

3.1.3. Preprocessing of text

```
In [0]: custom_words = ["fig", "figure", "et", "al", "al.", "also",
                        "data", "analyze", "study", "table", "using",
                        "method", "result", "conclusion", "author",
                        "find", "found", "show", '"', "’", "“", "”","cell","mutation"]

        stop_words = set(stopwords.words('english')+custom_words)
```

```
In [0]: # loading stop words from nltk library

        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()
            SnowballStemmer("english").stem(total_text)
            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
            return(total_text)
```

```
In [0]: from nltk.stem import *
        from nltk.stem.porter import *
        stemmer = PorterStemmer()
```

```
In [0]: for index, row in data_text.iterrows():
            if type(row['TEXT']) is str:

              total_text=row['TEXT']
```

```
In [52]: from nltk.tokenize import word_tokenize
         from nltk.stem import WordNetLemmatizer
         from nltk.corpus import stopwords
         import nltk
         nltk.download('punkt')
         nltk.download('stopwords')
         nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

```
Out[52]: True
```

Text is tokenized, cleaned of stopwords and lemmatized for word frequency analysis.

Tokenization obviously takes a lot of time on a corpus like this. So bear that in mind. May skip this, use a simpler tokenizer like ToktokTokenizer or just use str.split() instead.

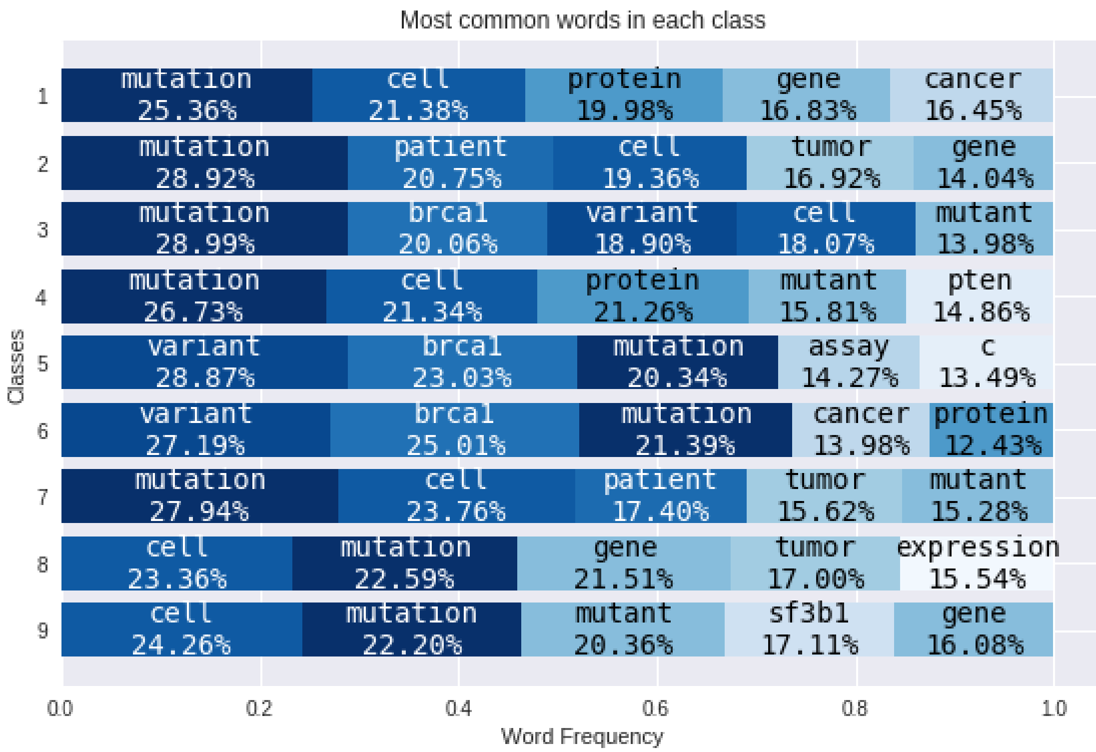
```
In [0]: https://www.kaggle.com/umutto/preliminary-data-analysis-using-word2vec
        wordnet_lemmatizer = WordNetLemmatizer()
```

```
class_corpus = result.groupby('Class').apply(lambda x: x['TEXT'].str.cat())
class_corpus = class_corpus.apply(lambda x: Counter(
    [wordnet_lemmatizer.lemmatize(w)
     for w in word_tokenize(x)
     if w.lower() not in stop_words and not w.isdigit()]
))
```

Lets look at the dominant words in classes. And see if we can find any correlation.

```
In [54]: class_freq = class_corpus.apply(lambda x: x.most_common(5))
         class_freq = pd.DataFrame.from_records(class_freq.values.tolist()).set_index(class_freq

         def normalize_row(x):
             label, repetition = zip(*x)
             t = sum(repetition)
             r = [n/t for n in repetition]
             return list(zip(label,r))

         class_freq = class_freq.apply(lambda x: normalize_row(x), axis=1)

         # set unique colors for each word so it's easier to read
         all_labels = [x for x in class_freq.sum().sum() if isinstance(x,str)]
         unique_labels = set(all_labels)
         cm = plt.get_cmap('Blues_r', len(all_labels))
         colors = {k:cm(all_labels.index(k)/len(all_labels)) for k in all_labels}

         fig, ax = plt.subplots()

         offset = np.zeros(9)
         for r in class_freq.iteritems():
             label, repetition = zip(*r[1])
             ax.barh(range(len(class_freq)), repetition, left=offset, color=[colors[l] for l in
             offset += repetition

         ax.set_yticks(np.arange(len(class_freq)))
         ax.set_yticklabels(class_freq.index)
         ax.invert_yaxis()

         # annotate words
         offset_x = np.zeros(9)
         for idx, a in enumerate(ax.patches):
             fc = 'k' if sum(a.get_fc()) > 2.5 else 'w'
             ax.text(offset_x[idx%9] + a.get_width()/2, a.get_y() + a.get_height()/2,
                     '{}\n{:.2%}'.format(all_labels[idx], a.get_width()),
                     ha='center', va='center', color=fc, fontsize=14, family='monospace')
             offset_x[idx%9] += a.get_width()
```

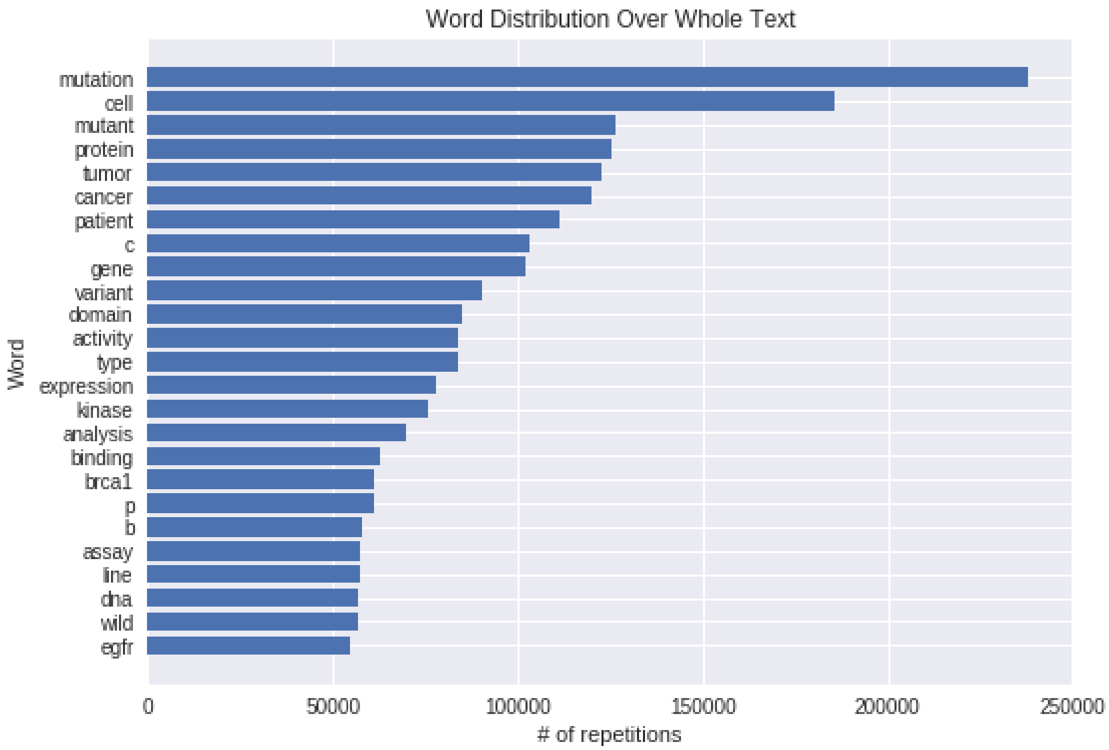
```
ax.set_title('Most common words in each class')
ax.set_xlabel('Word Frequency')
ax.set_ylabel('Classes')

plt.tight_layout()
plt.show()
```



Mutation and cell seems to be commonly dominating in all classes, not very informative. But the graph is still helpful. And would give more insight if we were to ignore most common words. Let's plot how many times 25 most common words appear in the whole corpus.

```
In [55]: whole_text_freq = class_corpus.sum()

         fig, ax = plt.subplots()

         label, repetition = zip(*whole_text_freq.most_common(25))

         ax.barh(range(len(label)), repetition, align='center')
         ax.set_yticks(np.arange(len(label)))
         ax.set_yticklabels(label)
         ax.invert_yaxis()

         ax.set_title('Word Distribution Over Whole Text')
         ax.set_xlabel('# of repetitions')
```

```
ax.set_ylabel('Word')

plt.tight_layout()
plt.show()
```




```
In [0]: #text processing stage.
        start_time = time.clock()
        for index, row in data_text.iterrows():
            if type(row['TEXT']) is str:
                nlp_preprocessing(row['TEXT'], index, 'TEXT')
            else:
                print("there is no text description for id:",index)
        print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 211.52816454299833 seconds
```

```
In [0]: #merging both gene_variations and text data based on ID
        result = pd.merge(data, data_text,on='ID', how='left')
        result.head()
```

```
In [0]: result[result.isnull().any(axis=1)]

Out[0]:         ID    Gene             Variation  Class TEXT
        1109  1109   FANCA                S1088F      1  NaN
        1277  1277  ARID5B  Truncating Mutations      1  NaN
        1407  1407   FGFR3                 K508M      6  NaN
        1639  1639    FLT1         Amplification      6  NaN
        2755  2755    BRAF                 G596C      7  NaN
```

```
In [0]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [0]: result['COMBINED']=result['TEXT'] +" "+result['Gene'] +' '+result['Variation']
```

```
In [0]: result[result['ID']==1109]

Out[0]:       Unnamed: 0    ID   Gene Variation  Class          TEXT  \
        1109        1109  1109  FANCA    S1088F      1  FANCA S1088F

                                 COMBINED  len
        1109  FANCA S1088F FANCA S1088F    2
```

```
In [39]: result.head()

Out[39]:    Unnamed: 0  ID    Gene             Variation  Class  \
         0           0   0  FAM58A  Truncating Mutations      1
         1           1   1     CBL                 W802*      2
         2           2   2     CBL                 Q249E      2
         3           3   3     CBL                 N454D      3
         4           4   4     CBL                 L399V      4

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

                                                COMBINED   len
         0  FAM58A Truncating Mutations cyclin dependent k...  4215
         1  CBL W802* abstract background non small lung c...  3978
         2  CBL Q249E abstract background non small lung c...  3978
         3  CBL N454D recent evidence demonstrated acquire...  3720
         4  CBL L399V oncogenic mutations monomeric casita...  4092
```

3.1.4. Test, Train and Cross Validation Split

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

```
In [0]: y_true = result['Class'].values
        result.Gene      = result.Gene.str.replace('\s+', '_')
```

```
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

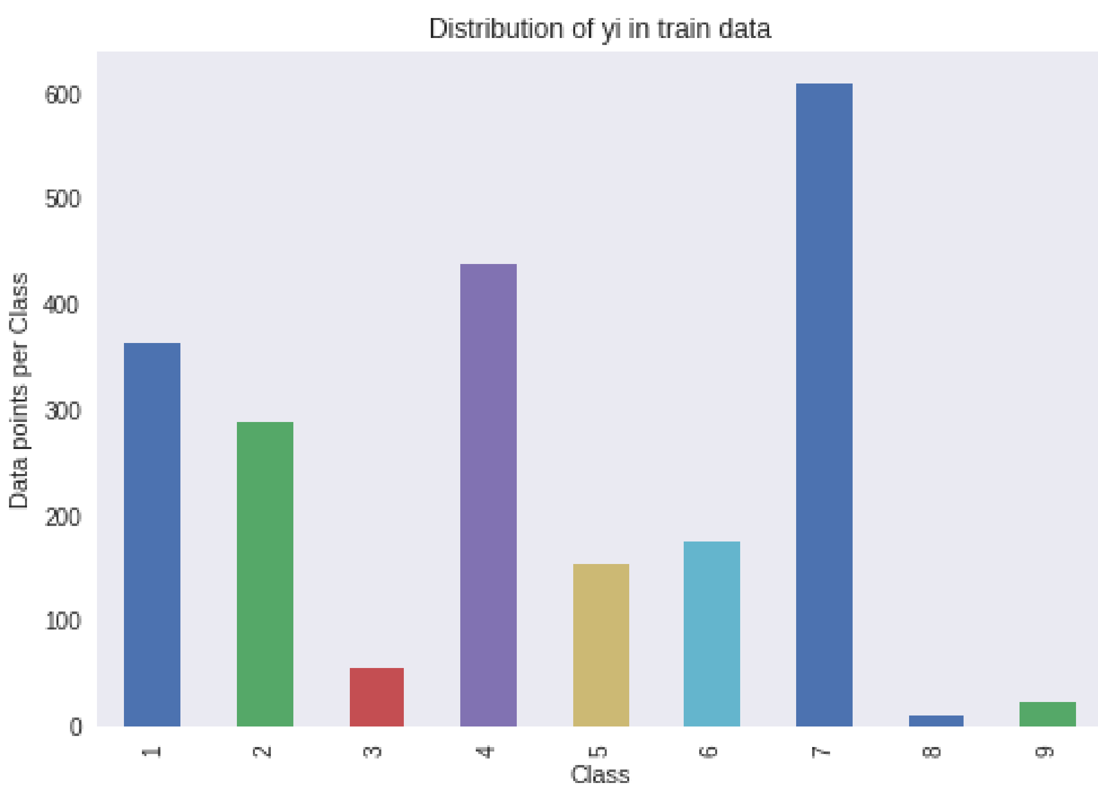
```
In [0]: print('Number of data points in train data:', train_df.shape[0])
        print('Number of data points in test data:', test_df.shape[0])
        print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

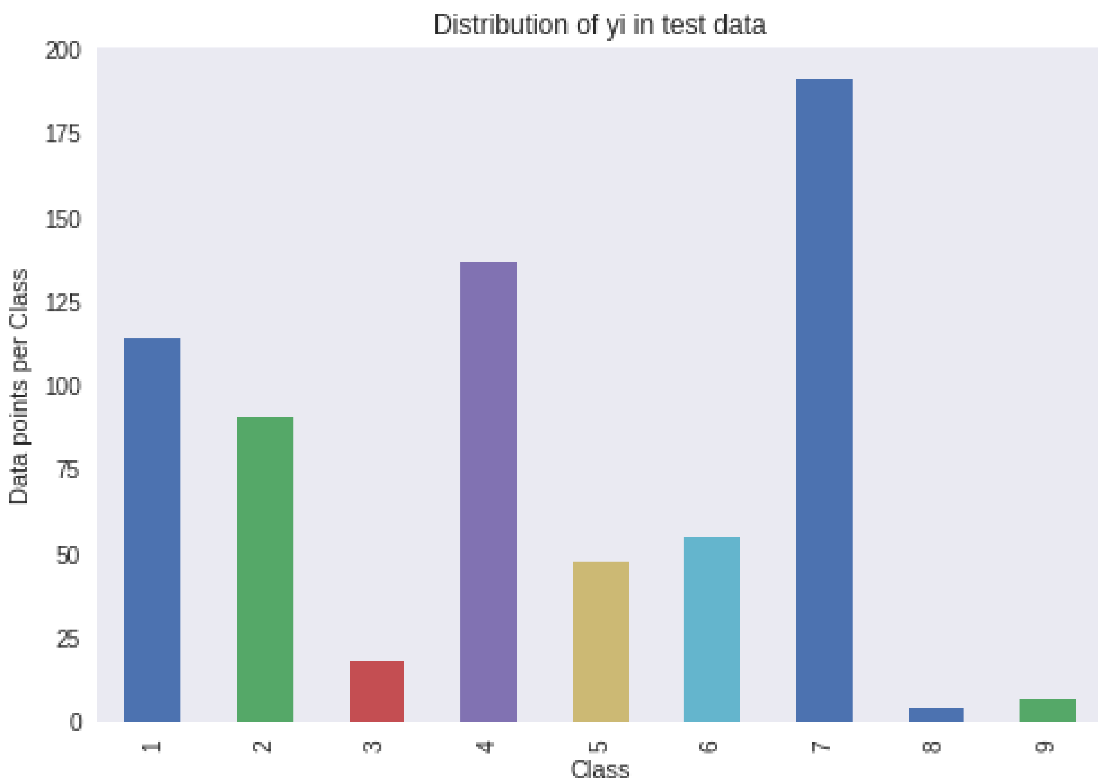
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [0]: # it returns a dict, keys as class labels and values as the number of data points in tha
        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],


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

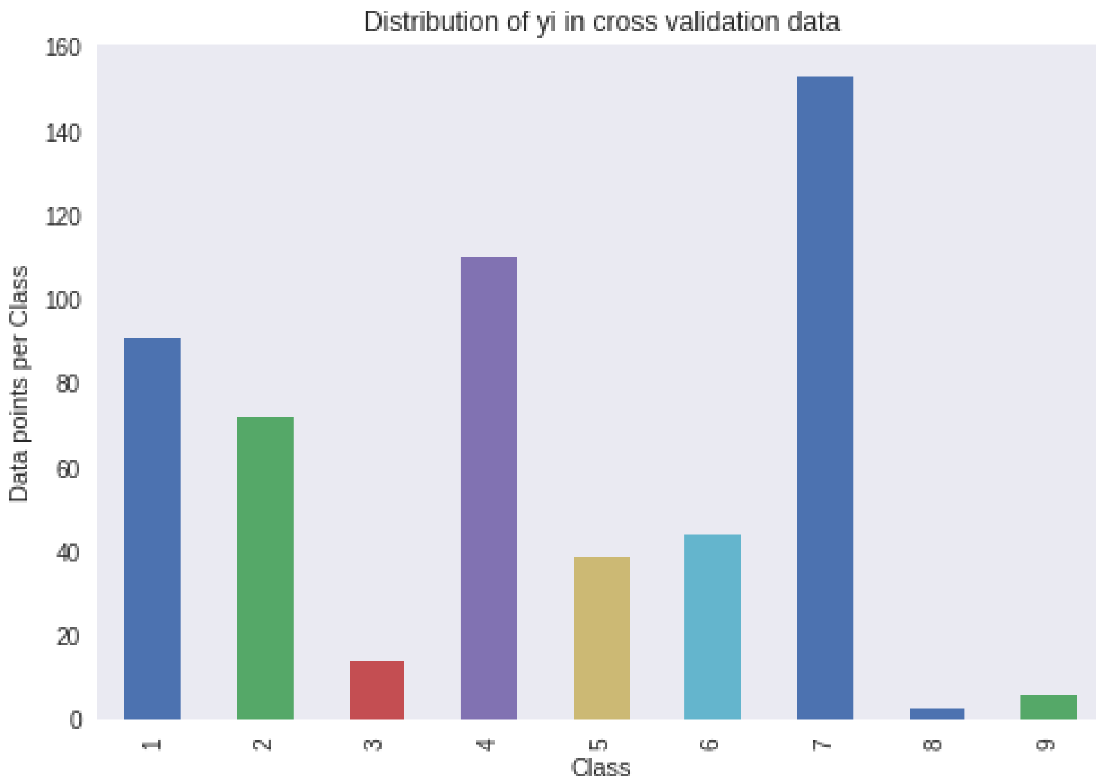
```
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1,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

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.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], '(
```



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




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




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

3.2 Prediction using a 'Random' Model

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

```
In [0]: # This function plots the confusion matrices given y_i, y_i_hat.
        def plot_confusion_matrix(test_y, predict_y):
            C = confusion_matrix(test_y, predict_y)
            # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predict

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

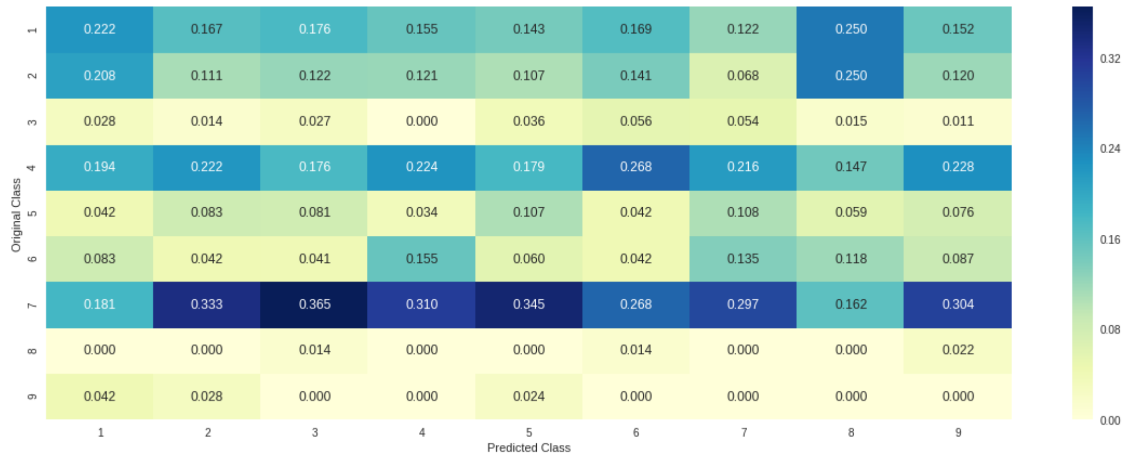
```
# C = [[1, 2],
#      [3, 4]]
# C.T = [[1, 3],
#        [2, 4]]
# C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in tu
# C.sum(axix =1) = [[3, 7]]
# ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
#                            [2/3, 4/7]]

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

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

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

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

In [0]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
```

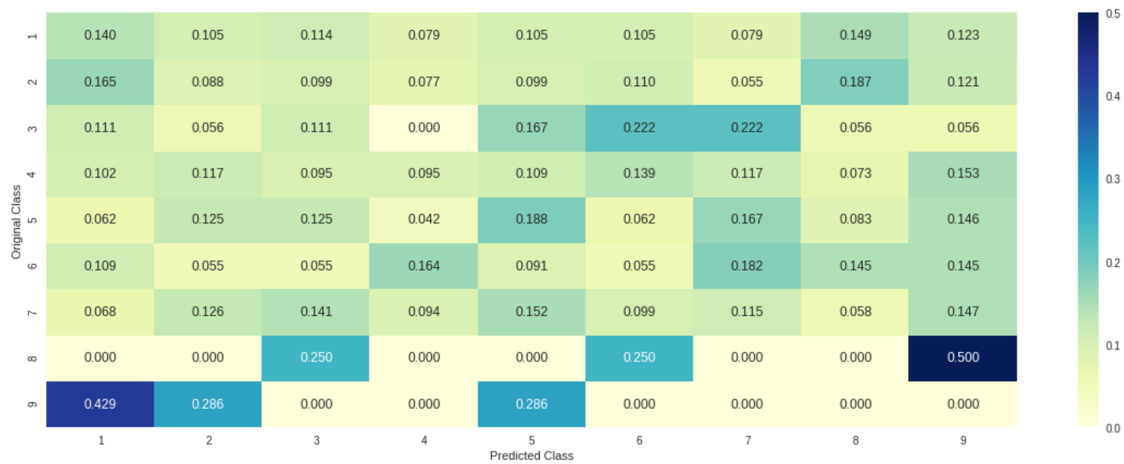
```
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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_


# 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=1

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

```
Log loss on Cross Validation Data using Random Model 2.451281904911502
Log loss on Test Data using Random Model 2.521993069687281
-------------------- Confusion matrix --------------------
```


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



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



3.3 Univariate Analysis

```
In [0]: # code for response coding with Laplace smoothing.
        # alpha : used for laplace smoothing
        # feature: ['gene', 'variation']
        # df: ['train_df', 'test_df', 'cv_df']
        # algorithm
        # ----------
        # Consider all unique values and the number of occurances of given feature in train data
        # build a vector (1*9) , the first element = (number of times it occured in class1 + 10*
        # gv_dict is like a look up table, for every gene it store a (1*9) representation of it
        # for a value of feature in df:
        # if it is in train data:
```

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

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

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

    # denominator will contain the number of time that particular feature occured in who
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticu
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation   Class
```

```
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particula
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181818
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

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

```
Number of Unique Genes : 233
BRCA1     178
TP53      107
BRCA2      84
EGFR       82
PTEN       73
KIT        66
BRAF       51
ERBB2      44
ALK        44
PDGFRA     39
Name: Gene, dtype: int64
```

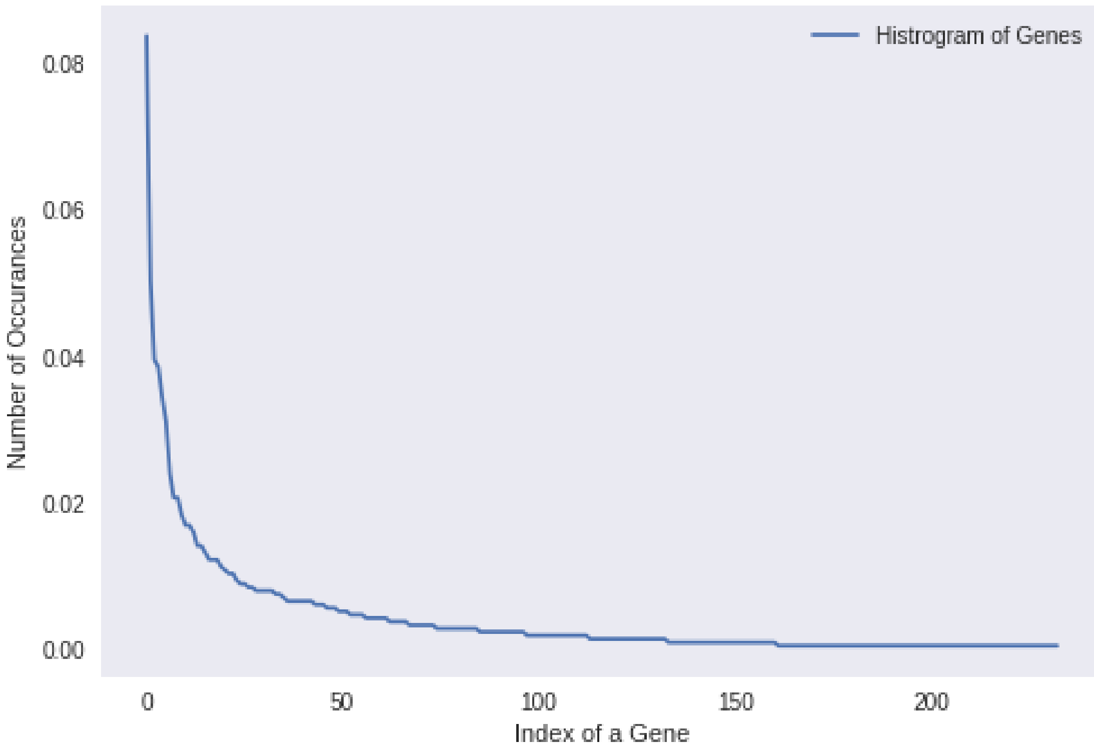
```
In [0]: unique_len = train_df['len'].value_counts()
        print('Number of Unique Genes :', unique_len.shape[0])
        # the top 10 genes that occured most
        print(unique_len.head(10))
```

```
Number of Unique Genes : 1286
6102    30
4357    28
3390    24
4753    24
4425    20
2967    17
3679    17
2264    17
2946    13
3429    12
Name: len, dtype: int64
```

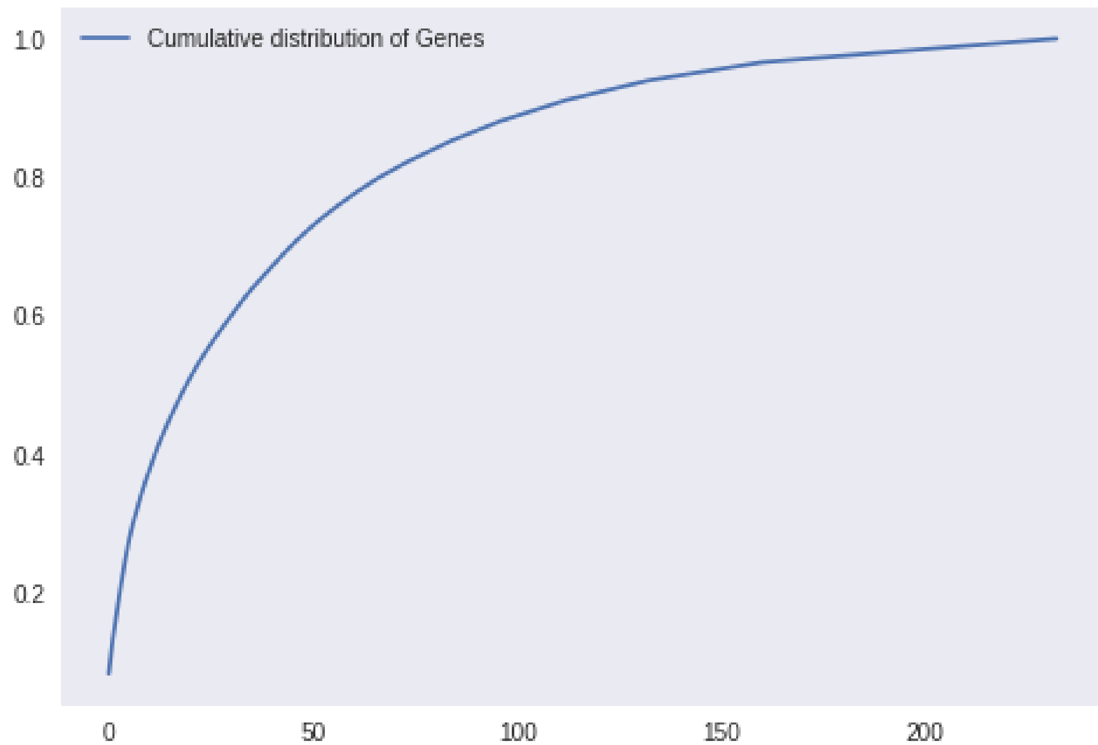
```
In [0]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the tra
```

```
Ans: There are 233 different categories of genes in the train data, and they are distibuted as f
```

In [0]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [0]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

One hot Encoding

Response coding

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

```
In [0]: #response-coding of the Gene feature
        # alpha is used for laplace smoothing
        alpha = 1
        # train gene feature
        train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
        # test gene feature
        test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
        # cross validation gene feature
        cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

```
In [0]: #response-coding of the len feature
        # alpha is used for laplace smoothing
        alpha = 1
        # train len feature
```

```
train_len_feature_responseCoding = np.array(get_gv_feature(alpha, "len", train_df))
# test len feature
test_len_feature_responseCoding = np.array(get_gv_feature(alpha, "len", test_df))
# cross validation len feature
cv_len_feature_responseCoding = np.array(get_gv_feature(alpha, "len", cv_df))
```

In [0]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding metho
```

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

In [0]:
```
# one-hot encoding of len feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [0]:
```
cv_len_feature_onehotCoding=cv_len_feature_responseCoding
test_len_feature_onehotCoding=test_len_feature_responseCoding
train_len_feature_onehotCoding=train_len_feature_responseCoding
```

In [0]:
```
%%javascript
const listenerChannel = new BroadcastChannel('channel');
listenerChannel.onmessage = (msg) => {
  const div = document.createElement('div');
  div.textContent = msg.data;
  document.body.appendChild(div);
};
```

In [0]:
```
%%javascript
const senderChannel = new BroadcastChannel('channel');
senderChannel.postMessage('Hello world!');
```

In [0]:
```
train_df['len'].head()
```

Out[0]:
```
745      ERBB2
1148       MET
2111       B2M
719      ERBB2
2928    NFE2L2
Name: Gene, dtype: object
```

In [0]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding metho
```

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

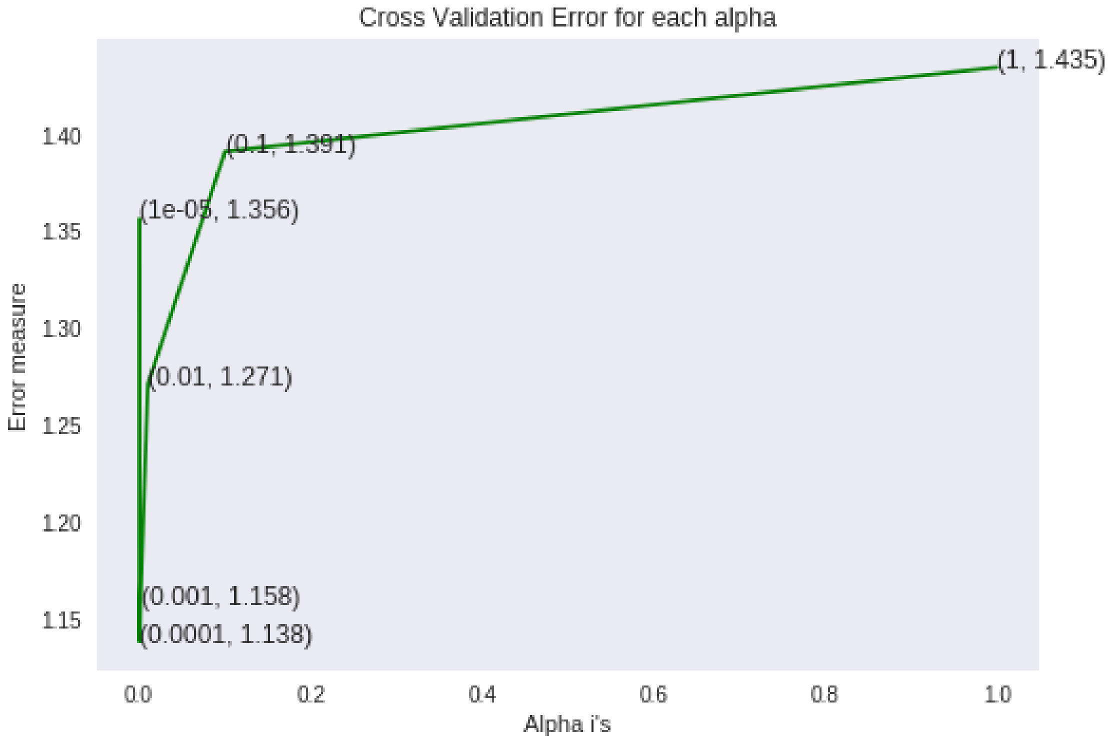
Q4. How good is this gene feature in predicting y_i?

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 len feature (one hot encoded) to predict y_i.

In [0]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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_los
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
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
```

```
For values of alpha =  1e-05 The log loss is: 1.3564126143908624
For values of alpha =  0.0001 The log loss is: 1.1382271621095041
For values of alpha =  0.001 The log loss is: 1.1579492351082075
For values of alpha =  0.01 The log loss is: 1.2711884486932745
For values of alpha =  0.1 The log loss is: 1.3911791917502785
For values of alpha =  1 The log loss is: 1.4345970017263474
```



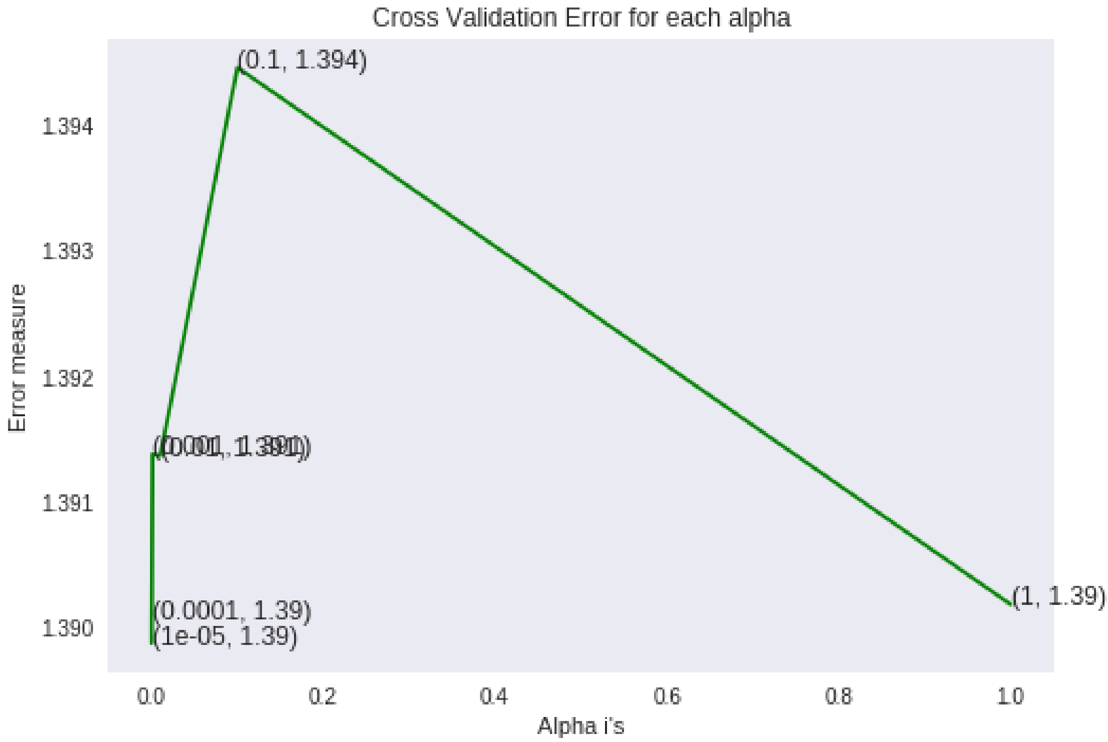

```
For values of best alpha =  0.0001 The train log loss is: 1.0457496698652227
For values of best alpha =  0.0001 The cross validation log loss is: 1.1382271621095041
For values of best alpha =  0.0001 The test log loss is: 1.2273574455829162
```

```
In [0]: alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

        # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
        # ------------------------------
        # default parameters
        # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
        # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
```

```python
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_len_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_len_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_len_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, labe

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_len_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_len_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_len_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_los
predict_y = sig_clf.predict_proba(cv_len_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
predict_y = sig_clf.predict_proba(test_len_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
```

```
For values of alpha =  1e-05 The log loss is: 1.3898777821619115
For values of alpha =  0.0001 The log loss is: 1.39007892299594
For values of alpha =  0.001 The log loss is: 1.391384755877006
```

```
For values of alpha =  0.01 The log loss is: 1.391369045090057
For values of alpha =  0.1 The log loss is: 1.394460558124926
For values of alpha =  1 The log loss is: 1.3901878296541657
```




```
For values of best alpha =  1e-05 The train log loss is: 0.6556955364430529
For values of best alpha =  1e-05 The cross validation log loss is: 1.3898777821619115
For values of best alpha =  1e-05 The test log loss is: 1.4131877555947476
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

In [0]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_gen

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverag
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cover
```

```
Q6. How many data points in Test and CV datasets are covered by the  233  genes in train dataset
Ans
1. In test data 650 out of 665 : 97.74436090225564
```

```
2. In cross validation data 509 out of  532 : 95.67669172932331
```

3.2.2 Univariate Analysis on Variation Feature
Q7. Variation, What type of feature is it ?
Ans. Variation is a categorical variable
Q8. How many categories are there?

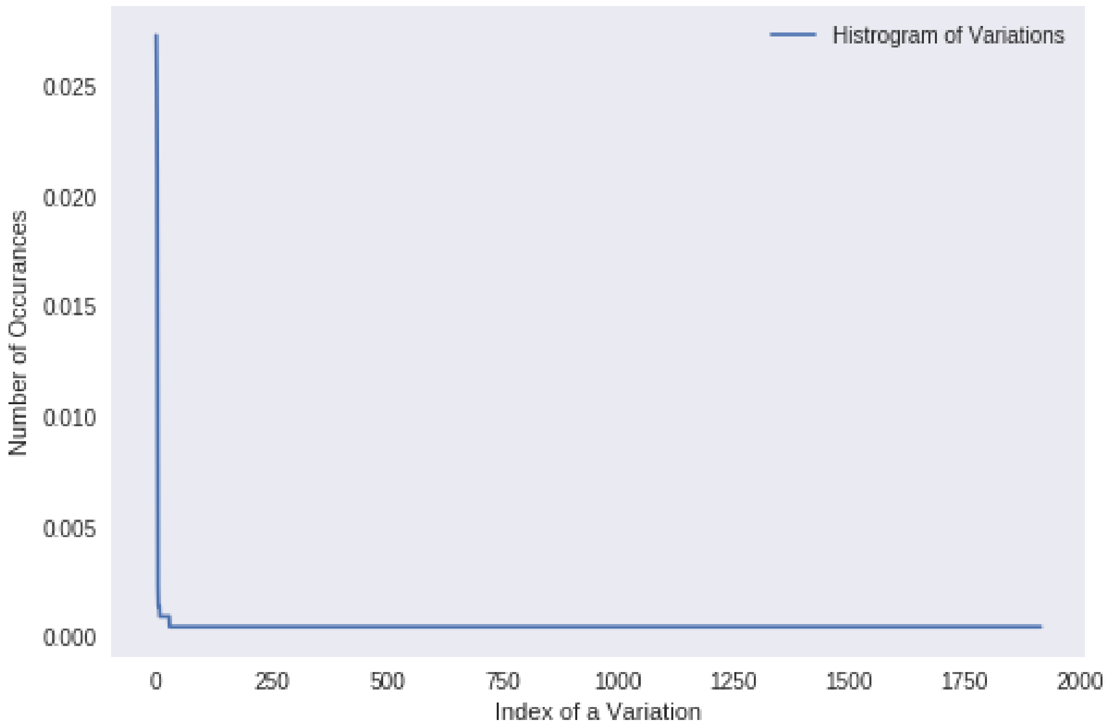
```
In [0]: unique_variations = train_df['Variation'].value_counts()
        print('Number of Unique Variations :', unique_variations.shape[0])
        # the top 10 variations that occured most
        print(unique_variations.head(10))
```

```
Number of Unique Variations : 1915
Truncating_Mutations    58
Amplification           53
Deletion                45
Fusions                 26
Overexpression           5
T58I                     3
E17K                     3
G12V                     3
Q61L                     2
P130S                    2
Name: Variation, dtype: int64
```

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

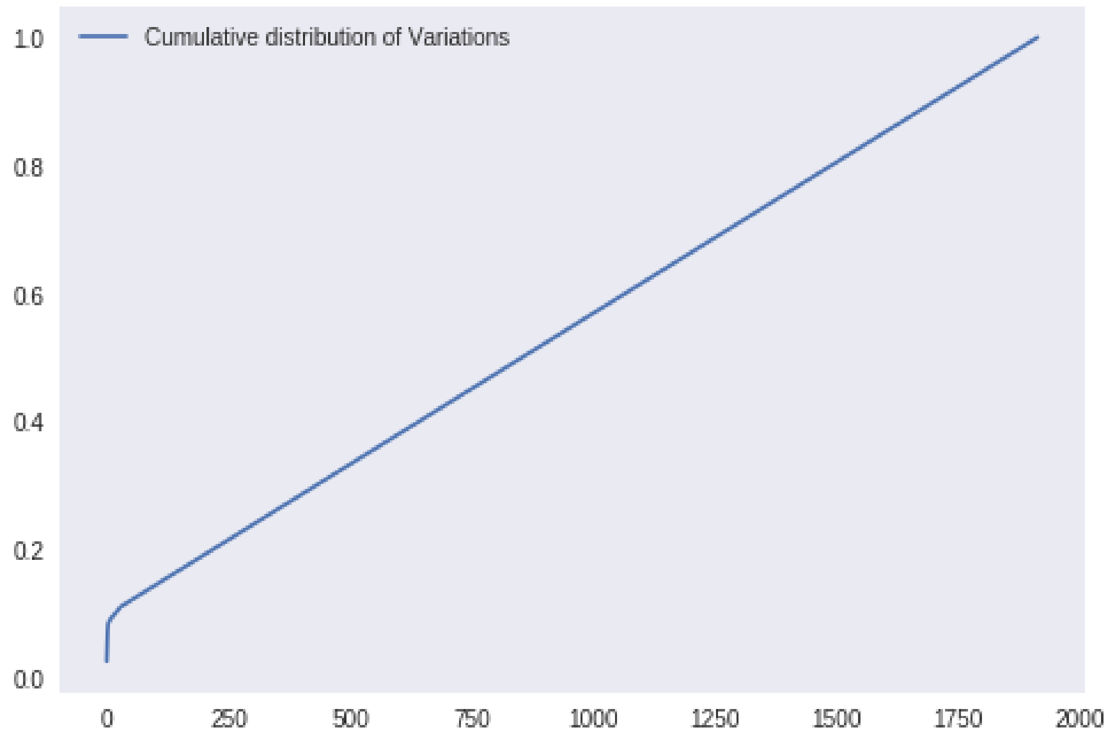
```
Ans: There are 1915 different categories of variations in the train data, and they are distibute
```

```
In [0]: s = sum(unique_variations.values);
        h = unique_variations.values/s;
        plt.plot(h, label="Histrogram of Variations")
        plt.xlabel('Index of a Variation')
        plt.ylabel('Number of Occurances')
        plt.legend()
        plt.grid()
        plt.show()
```




```
In [0]: c = np.cumsum(h)
        print(c)
        plt.plot(c,label='Cumulative distribution of Variations')
        plt.grid()
        plt.legend()
        plt.show()
```

```
[0.02730697 0.05225989 0.07344633 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

One hot Encoding

Response coding

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

```
In [0]: # alpha is used for laplace smoothing
        alpha = 1
        # train gene feature
        train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tra
        # test gene feature
        test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test
        # cross validation gene feature
        cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df)
```

```
In [0]: print("train_variation_feature_responseCoding is a converted feature using the response
```

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

```
In [0]: # one-hot encoding of variation feature.
        variation_vectorizer = TfidfVectorizer()
        train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Vari
```

```
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [0]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot enc
```

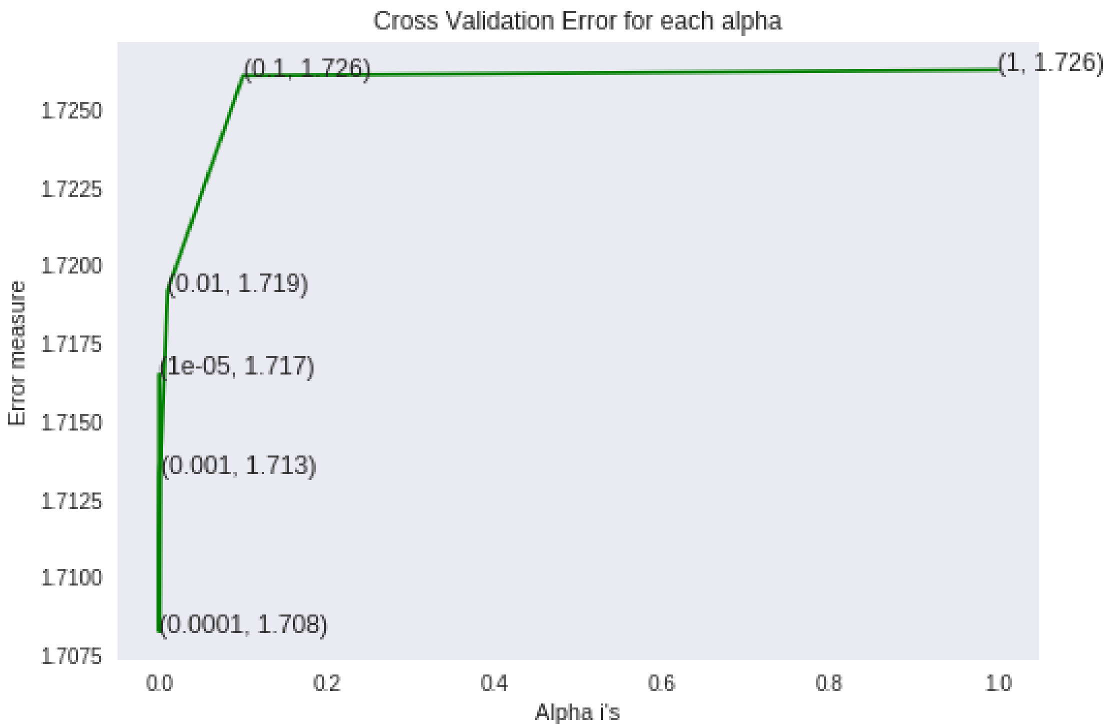
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. T
```

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

In [0]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

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_los
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
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
```

```
For values of alpha =  1e-05 The log loss is: 1.7165039488252285
For values of alpha =  0.0001 The log loss is: 1.7082690288902944
For values of alpha =  0.001 The log loss is: 1.7133399121685127
For values of alpha =  0.01 The log loss is: 1.7191902960065553
For values of alpha =  0.1 The log loss is: 1.7260700262600646
For values of alpha =  1 The log loss is: 1.7262611717838263
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7806552183900988
For values of best alpha =  0.0001 The cross validation log loss is: 1.7082690288902944
For values of best alpha =  0.0001 The test log loss is: 1.7182247449544625
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Not sure! But lets be very sure using the below analysis.

```
In [0]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], " g
        test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape
        cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
        print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverag
        print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cover
```

```
Q12. How many data points are covered by total  1915  genes in test and cross validation data se
Ans
1. In test data 56 out of 665 : 8.421052631578947
2. In cross validation data 55 out of  532 : 10.338345864661653
```

3.2.3 Univariate Analysis on Text Feature

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

```
In [0]: # cls_text is a data frame
        # for every row in data fram consider the 'TEXT'
        # split the words by space
        # make a dict with those words
        # increment its count whenever we see that word

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

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

```
            for word in row['COMBINED'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(wor
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['COMBI
            row_index += 1
    return text_feature_responseCoding
```

In [0]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train da
text_vectorizer = TfidfVectorizer(max_features=10000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['COMBINED'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numbe
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 10000
```

In [0]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [0]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
```

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

In [0]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feat
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_respo
```

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

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

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

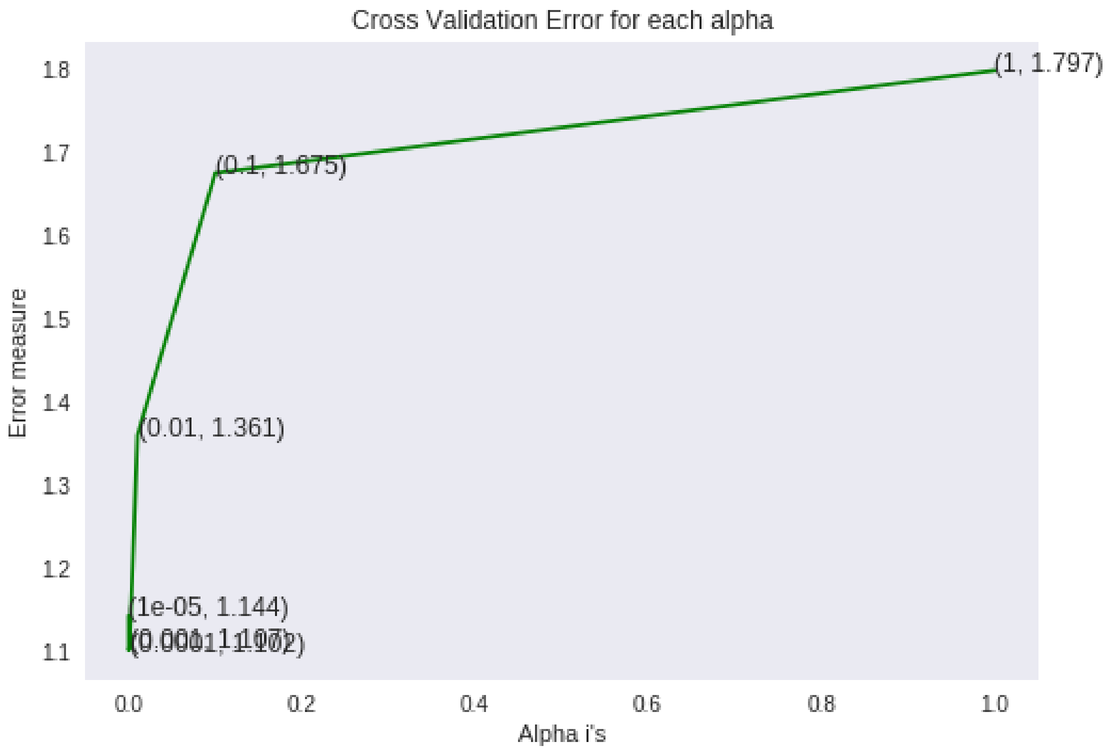
In [0]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=T
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

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

In [0]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot enc
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
```

```
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_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, labe

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

```
For values of alpha =  1e-05 The log loss is: 1.144428852696157
For values of alpha =  0.0001 The log loss is: 1.1022827860736473
For values of alpha =  0.001 The log loss is: 1.107231544016109
For values of alpha =  0.01 The log loss is: 1.3605679046131038
For values of alpha =  0.1 The log loss is: 1.6745531214112133
For values of alpha =  1 The log loss is: 1.7974558172371204
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7213801352123799
For values of best alpha =  0.0001 The cross validation log loss is: 1.1022827860736473
For values of best alpha =  0.0001 The test log loss is: 1.1137591620315554
```

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

In [0]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [0]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train dat
```

```
32.171 % of word of test data appeared in train data
36.948 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

```
In [0]: #Data preparation for ML models.

        #Misc. functionns for ML models


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

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

```
In [0]: def report_log_loss(train_x, train_y, test_x, test_y,  clf):
            clf.fit(train_x, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x, train_y)
            sig_clf_probs = sig_clf.predict_proba(test_x)
            return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

```
In [0]: # this function will be used just for naive bayes
        # for the given indices, we will print the name of the features
        # and we will check whether the feature present in the test point text or not
        def get_impfeature_names(indices, text, gene, var, no_features):
            gene_count_vec = TfidfVectorizer()
            var_count_vec = TfidfVectorizer()
            text_count_vec = TfidfVectorizer(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['COMBINED'])

            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
            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
            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

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

Stacking the three types of features

```
In [0]: # merging gene, variance and text features

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

        train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_fe
        test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
        cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one

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

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

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


        train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_varia
        test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variatio
        cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feat
```

```
train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_re
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_respo
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCod
```

```
In [0]: print("One hot encoding features :")
        print("(number of data points * number of features) in train data = ", train_x_onehotCod
        print("(number of data points * number of features) in test data = ", test_x_onehotCodin
        print("(number of data points * number of features) in cross validation data =", cv_x_on
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 11967)
(number of data points * number of features) in test data =  (665, 11967)
(number of data points * number of features) in cross validation data = (532, 11967)
```

```
In [0]: print(" Response encoding features :")
        print("(number of data points * number of features) in train data = ", train_x_responseC
        print("(number of data points * number of features) in test data = ", test_x_responseCod
        print("(number of data points * number of features) in cross validation data =", cv_x_re
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 36)
(number of data points * number of features) in test data =  (665, 36)
(number of data points * number of features) in cross validation data = (532, 36)
```

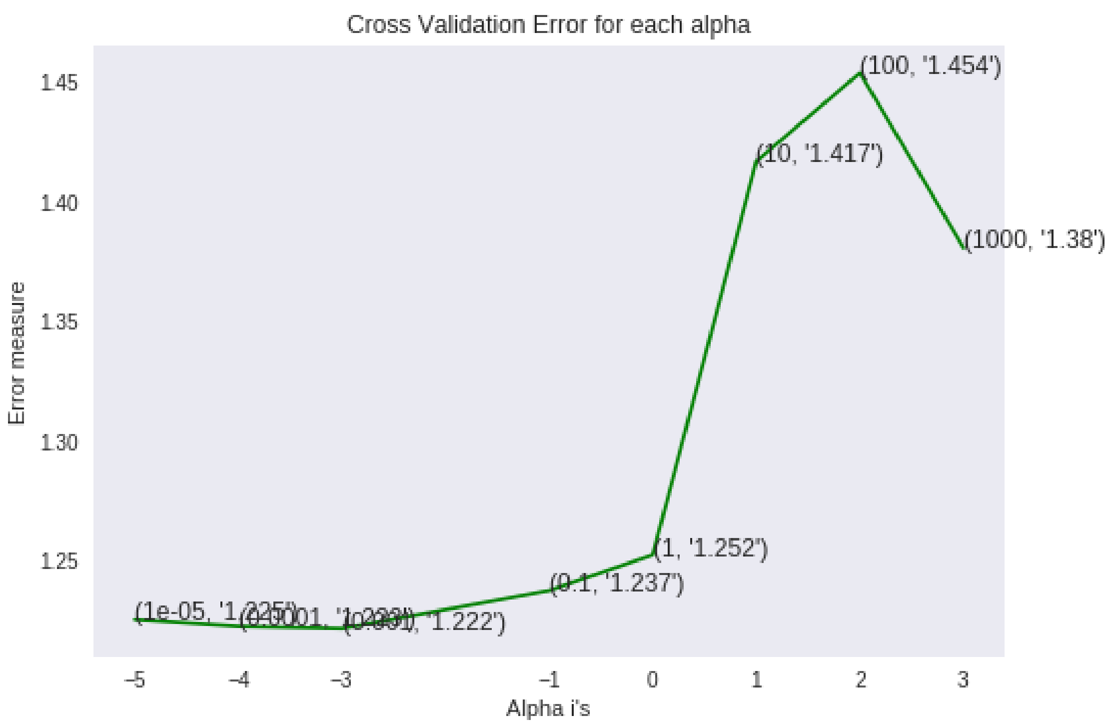
4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

```
In [0]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/mo
        # -------------------------
        # default paramters
        # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

        # some of methods of MultinomialNB()
        # fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X, y
        # predict(X)        Perform classification on an array of test vectors X.
        # predict_log_proba(X)        Return log-probability estimates for the test vector X.
        # -----------------------
        # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/na
        # -----------------------


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

```
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/na
#-----------------------


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-
    # to avoid rounding error while multiplying probabilites we use log-probability esti
    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_los
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
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
```

```
for alpha = 1e-05
Log Loss : 1.2253996626244668
for alpha = 0.0001
Log Loss : 1.2225450777016056
for alpha = 0.001
Log Loss : 1.2215988027794509
for alpha = 0.1
Log Loss : 1.2374676894937315
for alpha = 1
Log Loss : 1.2523703745397583
for alpha = 10
Log Loss : 1.4167177824993398
for alpha = 100
Log Loss : 1.453562941745203
for alpha = 1000
Log Loss : 1.3804835528424193
```




```
For values of best alpha =  0.001 The train log loss is: 0.7821438461253444
For values of best alpha =  0.001 The cross validation log loss is: 1.2215988027794509
For values of best alpha =  0.001 The test log loss is: 1.2527501546195547
```

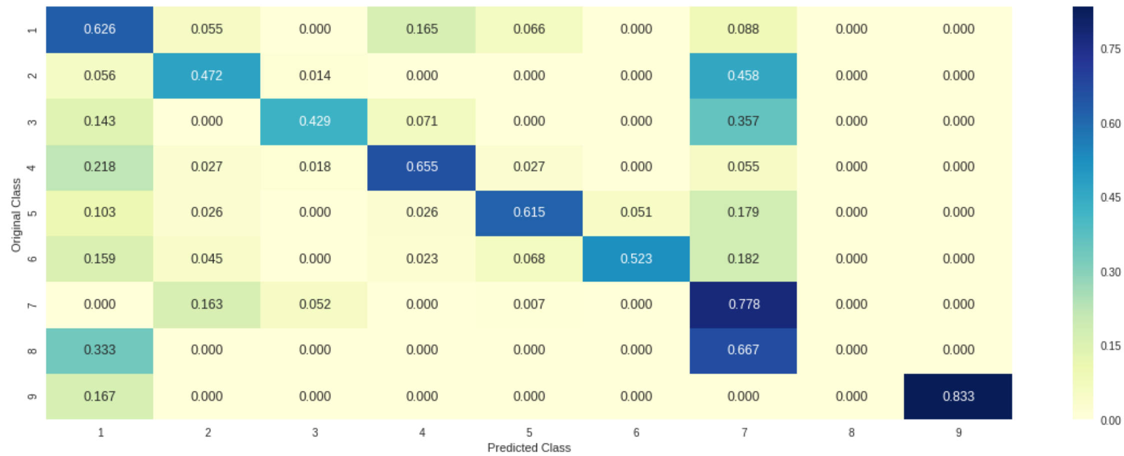
4.1.1.2. Testing the model with best hyper paramters

In [0]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/mo
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X, y
# predict(X)        Perform classification on an array of test vectors X.
# predict_log_proba(X)        Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/na
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# 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
# ----------------------------

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 estimate
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.2215988027794509
Number of missclassified point : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

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

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



4.1.1.3. Feature Importance, Incorrectly classified point

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2597 0.0569 0.0098 0.0925 0.0366 0.4454 0.0906 0.0046 0.0039]
Actual Class : 5
--------------------------------------------------
25 Text feature [18] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.1.1.4. Feature Importance, correctly classified point

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0858 0.5872 0.0121 0.1114 0.0433 0.042  0.108  0.0055 0.0047]
Actual Class : 2
--------------------------------------------------
26 Text feature [2013] present in test data point [True]
45 Text feature [agreement] present in test data point [True]
69 Text feature [437] present in test data point [True]
76 Text feature [1xkk] present in test data point [True]
Out of the top  100  features  4 are present in query point
```

4.2. K Nearest Neighbour Classification

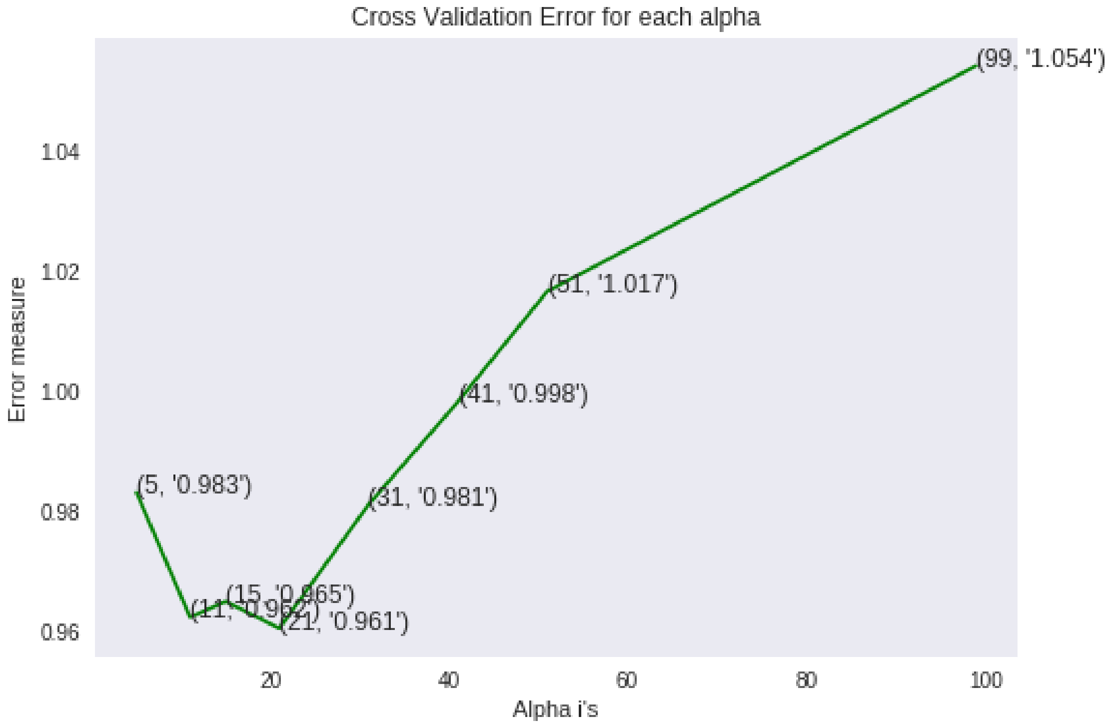
4.2.1. Hyper parameter tuning

In [0]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/gen
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
```

```
        clf = KNeighborsClassifier(n_neighbors=i)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-
        # to avoid rounding error while multiplying probabilites we use log-probability esti
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 0.9832047455945698
for alpha = 11
Log Loss : 0.9624396269483809
for alpha = 15
Log Loss : 0.9650523548498324
for alpha = 21
Log Loss : 0.9605821044765198
for alpha = 31
Log Loss : 0.9811975045589895
for alpha = 41
Log Loss : 0.9982955819201564
for alpha = 51
Log Loss : 1.0166756407029134
for alpha = 99
```

```
Log Loss : 1.0542034864451453
```

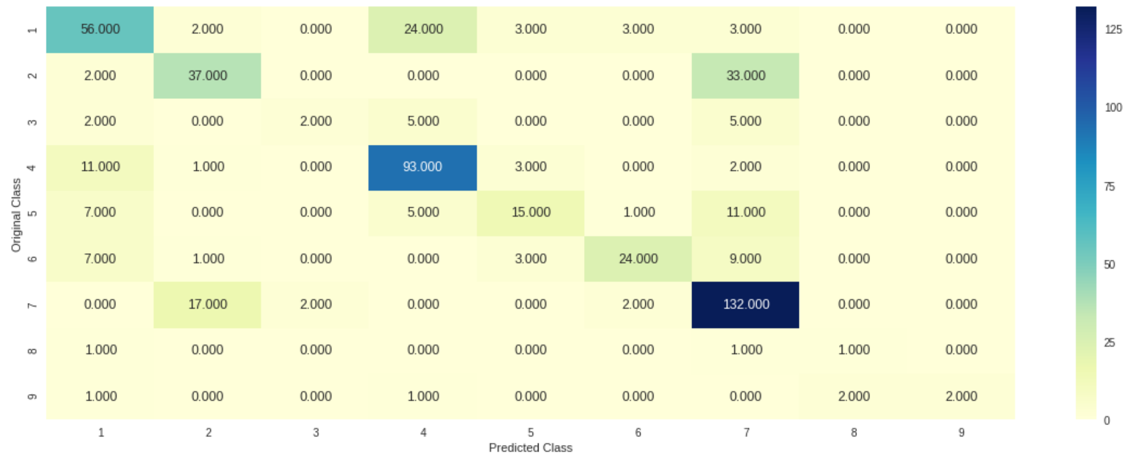



```
For values of best alpha =  21 The train log loss is: 0.6798003344279114
For values of best alpha =  21 The cross validation log loss is: 0.9605821044765198
For values of best alpha =  21 The test log loss is: 1.0406026837892697
```

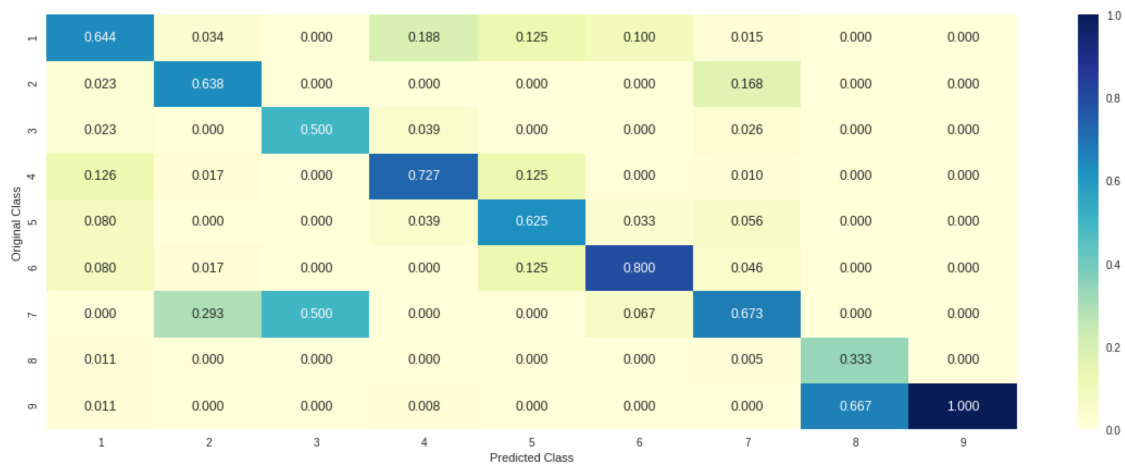
4.2.2. Testing the model with best hyper paramters

```
In [0]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/gen
        # -------------------------
        # default parameter
        # KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
        # metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

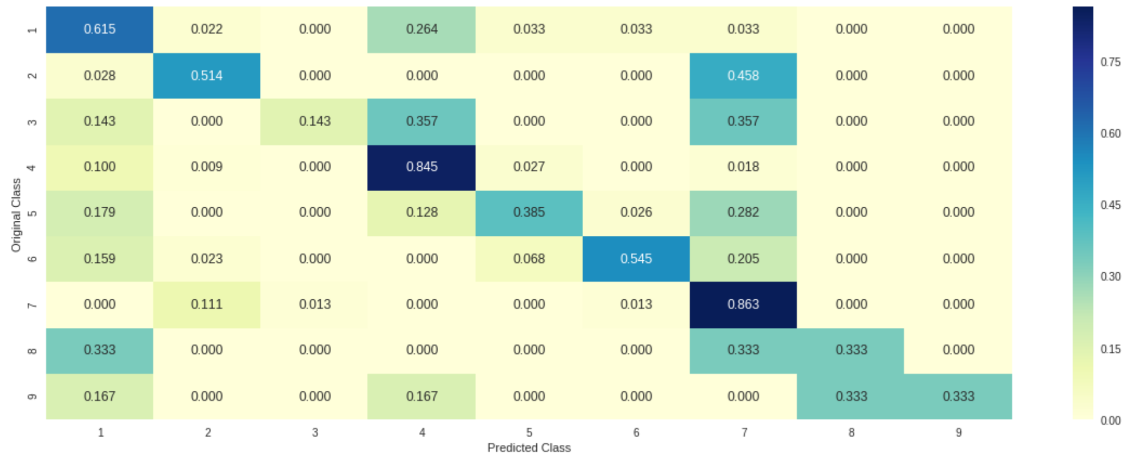
```
Log loss : 0.9605821044765198
Number of mis-classified points : 0.31954887218045114
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

```
In [0]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 7
Actual Class : 5
The  21  nearest neighbours of the test points belongs to classes [1 1 6 6 6 5 6 1 3 1 1 1 6 5 6
Fequency of nearest points : Counter({1: 11, 6: 6, 5: 2, 3: 1, 4: 1})
```

4.2.4. Sample Query Point-2

```
In [0]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)

        test_point_index = 11

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

```
        neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha
        print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test
        print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 21 and the nearest neighbours of the test points belongs to classes [7 7
Fequency of nearest points : Counter({7: 14, 1: 2, 2: 2, 5: 1, 3: 1, 6: 1})
```

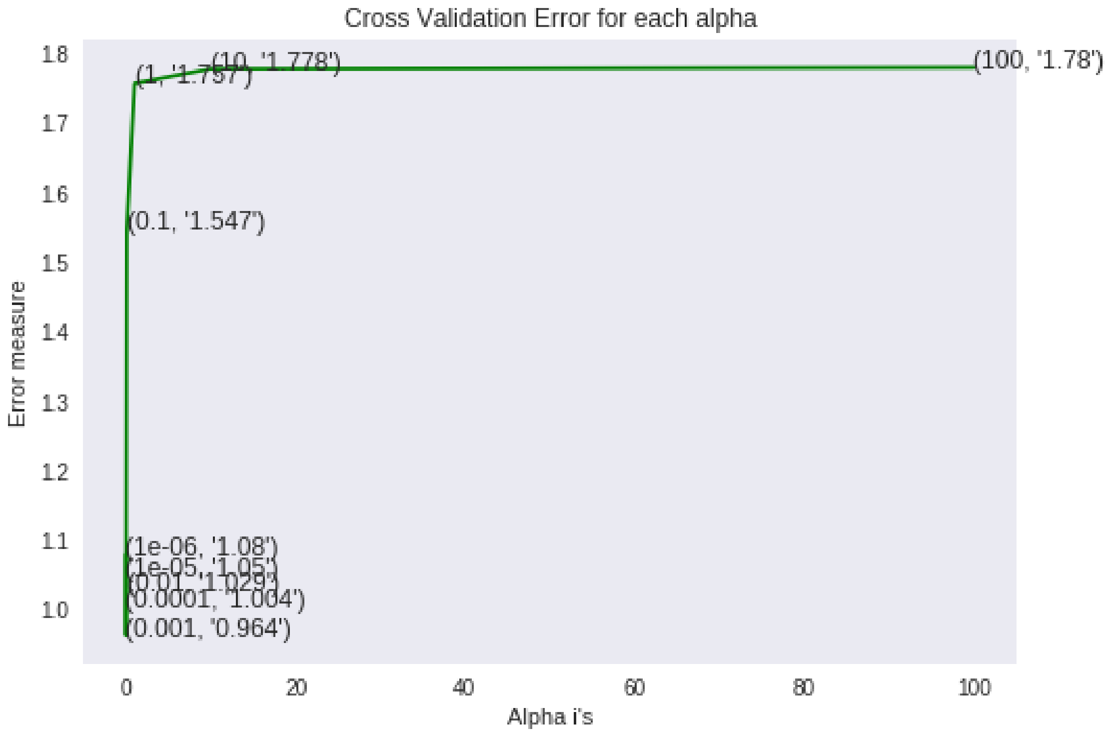
4.3. Logistic Regression
4.3.1. With Class balancing
4.3.1.1. Hyper paramter tuning

In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# 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)
```

```python
        clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
        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-
        # to avoid rounding error while multiplying probabilites we use log-probability esti
        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
    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_los
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
    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
```

```
for alpha = 1e-06
Log Loss : 1.080166709033941
for alpha = 1e-05
Log Loss : 1.0496434650830304
for alpha = 0.0001
Log Loss : 1.0041808225193936
for alpha = 0.001
Log Loss : 0.9635485640909678
for alpha = 0.01
Log Loss : 1.0291050474173005
for alpha = 0.1
Log Loss : 1.547163862488406
for alpha = 1
Log Loss : 1.7569572424701496
for alpha = 10
```

```
Log Loss : 1.7777267375456105
for alpha = 100
Log Loss : 1.7798702640500892
```




```
For values of best alpha =  0.001 The train log loss is: 0.547527934534752
For values of best alpha =  0.001 The cross validation log loss is: 0.9635485640909678
For values of best alpha =  0.001 The test log loss is: 1.0194900759147862
```

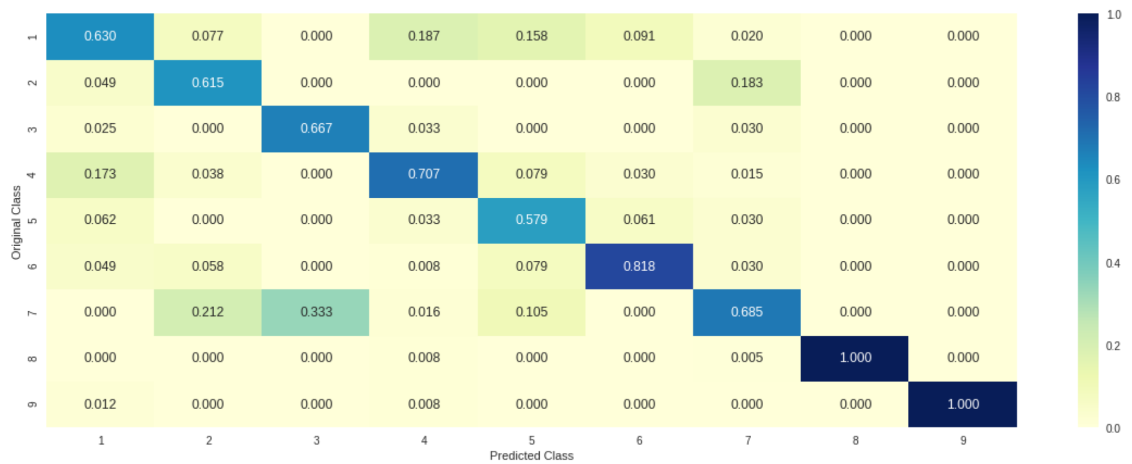
4.3.1.2. Testing the model with best hyper paramters

In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
```

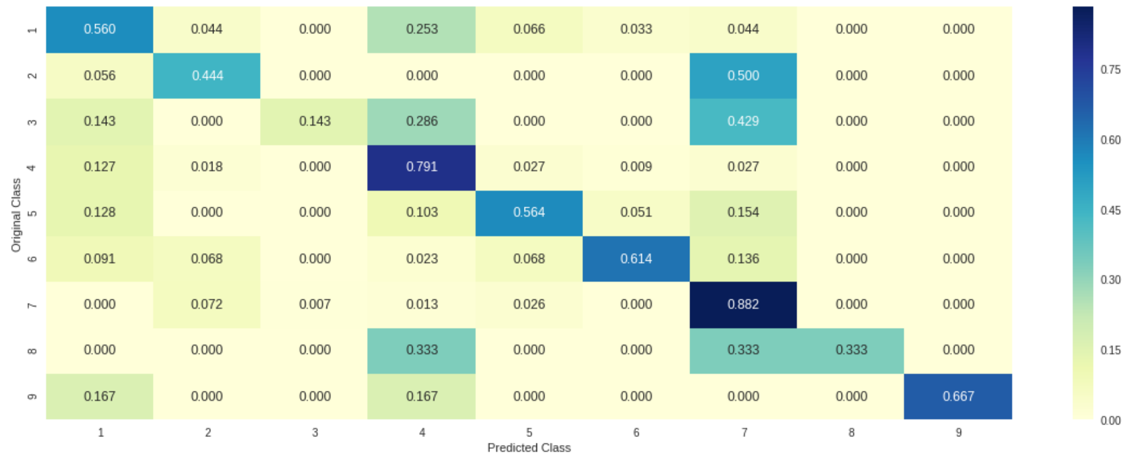
```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/ge
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y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

```
Log loss : 0.9635485640909678
Number of mis-classified points : 0.32142857142857145
-------------------- Confusion matrix --------------------
```


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




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



4.3.1.3. Feature Importance

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

4.3.1.3.1. Correctly Classified point

```
In [0]: # from tabulate import tabulate
        clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
        clf.fit(train_x_onehotCoding,train_y)
        test_point_index = 2
        no_feature = 500
        predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[1.400e-03 1.800e-03 6.300e-03 9.826e-01 4.200e-03 1.600e-03 5.0
  1.200e-03 3.000e-04]]
Actual Class : 4
--------------------------------------------------
34 Text feature [56] present in test data point [True]
72 Text feature [10q23] present in test data point [True]
363 Text feature [8547] present in test data point [True]
401 Text feature [30] present in test data point [True]
484 Text feature [a121p] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.3137 0.0082 0.0046 0.0296 0.0793 0.5523 0.0068 0.0037 0.0019]
Actual Class : 5
--------------------------------------------------
212 Text feature [1464] present in test data point [True]
213 Text feature [2006] present in test data point [True]
280 Text feature [bidentate] present in test data point [True]
302 Text feature [1842] present in test data point [True]
310 Text feature [2c] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.2. Without Class balancing

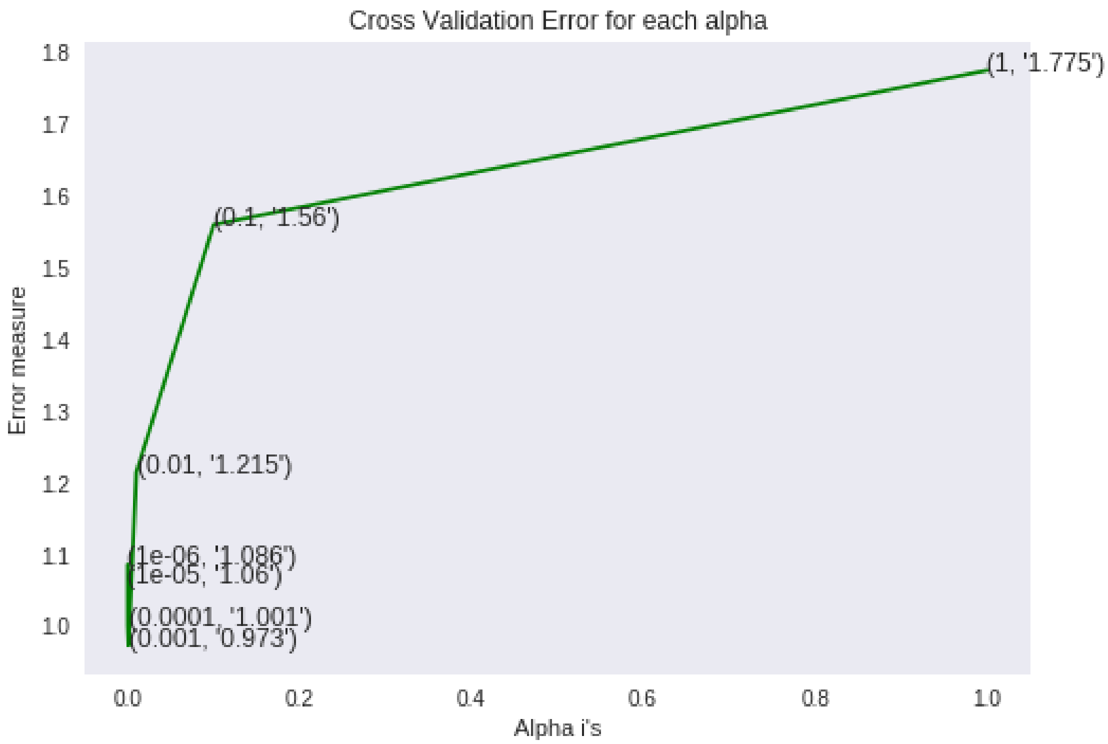
4.3.2.1. Hyper paramter tuning

In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# ----------------------------
# 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, 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-
    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()
```

```python
        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_los
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
        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
```

```
for alpha = 1e-06
Log Loss : 1.0860373034600141
for alpha = 1e-05
Log Loss : 1.0601009490823998
for alpha = 0.0001
Log Loss : 1.000763135814178
for alpha = 0.001
Log Loss : 0.9732001233997214
for alpha = 0.01
Log Loss : 1.2146823889723979
for alpha = 0.1
Log Loss : 1.5596874615954381
for alpha = 1
Log Loss : 1.774522471463847
```

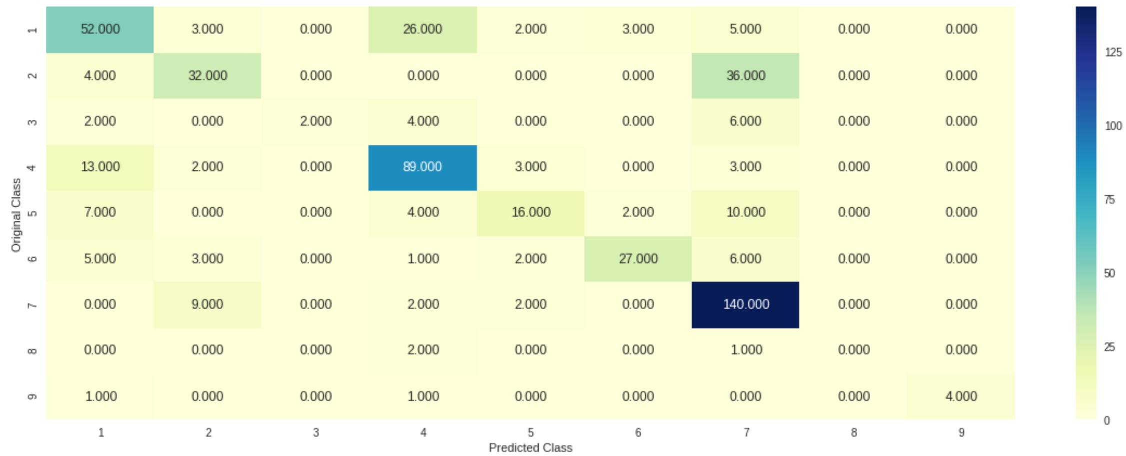



```
For values of best alpha =  0.001 The train log loss is: 0.5407828784168962
For values of best alpha =  0.001 The cross validation log loss is: 0.9732001233997214
For values of best alpha =  0.001 The test log loss is: 1.0259818467332322
```

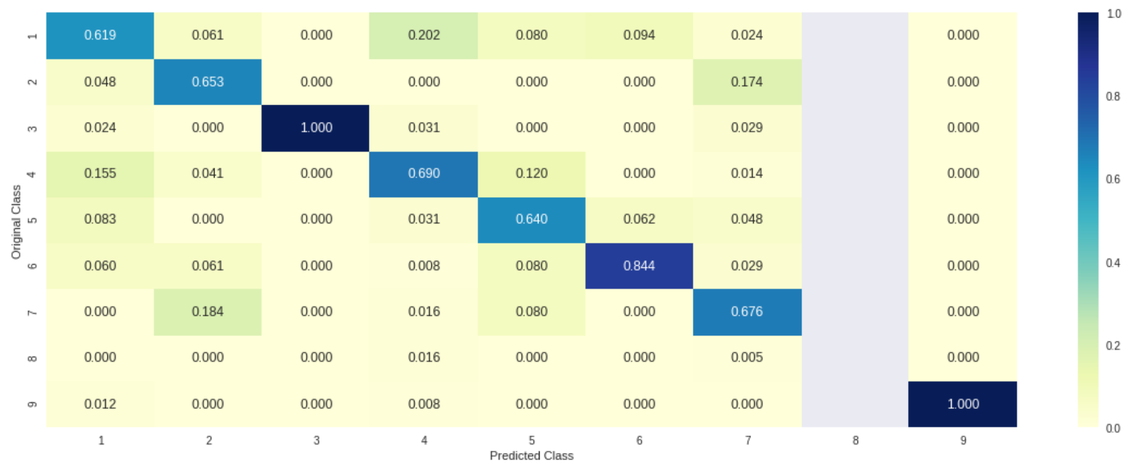
4.3.2.2. Testing model with best hyper parameters

```
In [0]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
        # ------------------------------
        # default parameters
        # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
        # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
        # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

        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
```

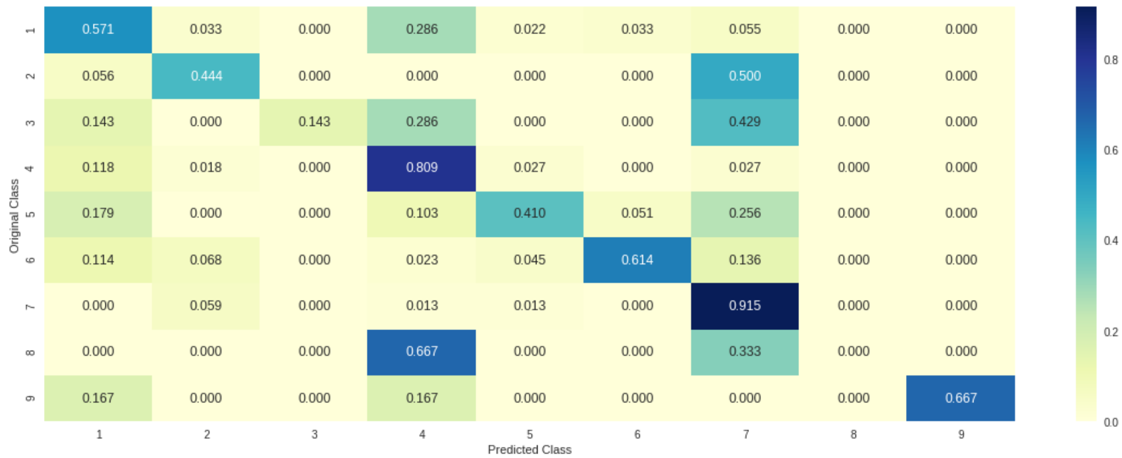
```
Log loss : 0.9732001233997214
Number of mis-classified points : 0.31954887218045114
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, incorrectly Classified point

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.3562 0.0087 0.0034 0.0285 0.0655 0.5248 0.0083 0.0032 0.0014]
Actual Class : 5
--------------------------------------------------
200 Text feature [2006] present in test data point [True]
210 Text feature [1464] present in test data point [True]
227 Text feature [bidentate] present in test data point [True]
299 Text feature [1842] present in test data point [True]
335 Text feature [2c] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.2.4. Feature Importance, Correctly Classified point

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

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

```
Predicted Class : 2
Predicted Class Probabilities: [[1.100e-03 7.232e-01 4.000e-04 3.000e-04 2.500e-03 6.400e-02 2.0
  5.000e-04 0.000e+00]]
Actual Class : 2
--------------------------------------------------
94 Text feature [badalian] present in test data point [True]
107 Text feature [9e11] present in test data point [True]
263 Text feature [018] present in test data point [True]
286 Text feature [2003] present in test data point [True]
299 Text feature [akt] present in test data point [True]
319 Text feature [1992] present in test data point [True]
335 Text feature [biopsies] present in test data point [True]
348 Text feature [anecdotal] present in test data point [True]
352 Text feature [another] present in test data point [True]
363 Text feature [april] present in test data point [True]
433 Text feature [acquire] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

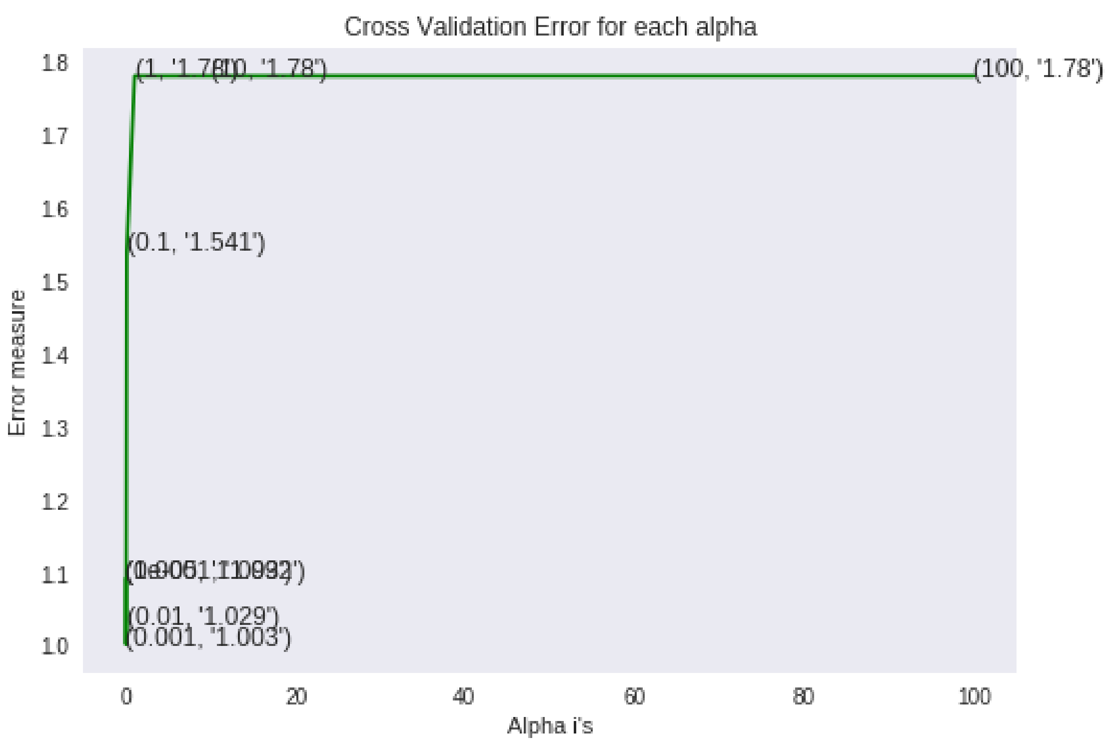
4.4. Linear Support Vector Machines
4.4.1. Hyper paramter tuning

```
In [0]: # read more about support vector machines with linear kernals here http://scikit-learn.c

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

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



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

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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', r
    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-
    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
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_los
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i
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
```

```
for C = 1e-05
```

```
Log Loss : 1.0928808735979871
for C = 0.0001
Log Loss : 1.092420364219074
for C = 0.001
Log Loss : 1.00253063506668
for C = 0.01
Log Loss : 1.0293643129990702
for C = 0.1
Log Loss : 1.540843023582192
for C = 1
Log Loss : 1.7801052554060288
for C = 10
Log Loss : 1.78012353130245
for C = 100
Log Loss : 1.780123506184307
```




```
For values of best alpha =  0.001 The train log loss is: 0.5239202735659587
For values of best alpha =  0.001 The cross validation log loss is: 1.00253063506668
For values of best alpha =  0.001 The test log loss is: 1.0716238450789997
```

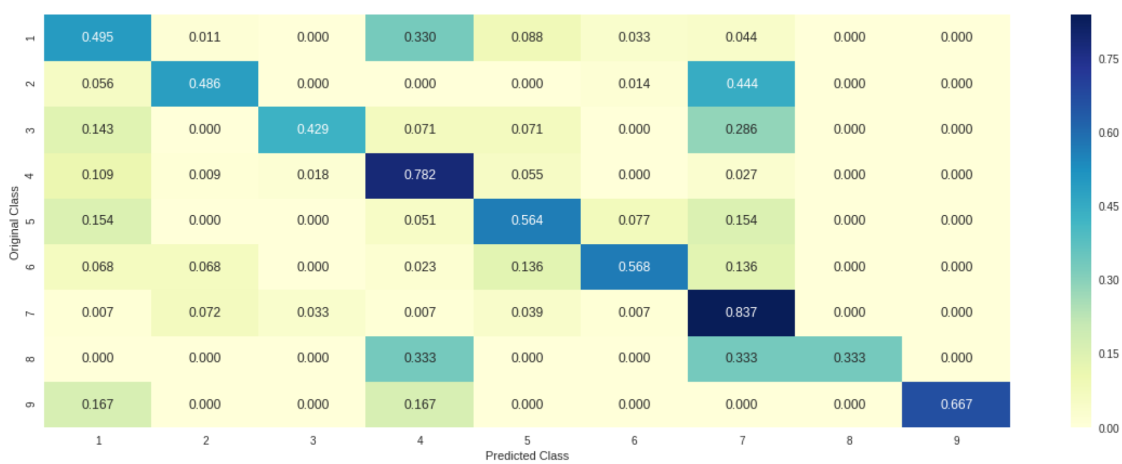
4.4.2. Testing model with best hyper parameters

```
In [0]: # read more about support vector machines with linear kernals here http://scikit-learn.c

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

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


        # 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
        predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y,
```

```
Log loss : 1.00253063506668
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```


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


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



4.3.3. Feature Importance
4.3.3.1. For incorrectly classified point

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

```
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1952 0.0308 0.0097 0.0426 0.1037 0.541  0.0699 0.0041 0.003 ]
Actual Class : 5
--------------------------------------------------
155 Text feature [2c] present in test data point [True]
210 Text feature [around] present in test data point [True]
257 Text feature [1842] present in test data point [True]
267 Text feature [1464] present in test data point [True]
435 Text feature [adopts] present in test data point [True]
466 Text feature [apparent] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.3.3.2. For correctly classified point

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0148 0.6434 0.0049 0.0043 0.0114 0.2234 0.0958 0.001  0.0011]
Actual Class : 2
--------------------------------------------------
8 Text feature [badalian] present in test data point [True]
210 Text feature [biopsies] present in test data point [True]
229 Text feature [2003] present in test data point [True]
240 Text feature [1995] present in test data point [True]
261 Text feature [752] present in test data point [True]
300 Text feature [9e11] present in test data point [True]
314 Text feature [1992] present in test data point [True]
316 Text feature [akt] present in test data point [True]
372 Text feature [adapt] present in test data point [True]
374 Text feature [april] present in test data point [True]
383 Text feature [anecdotal] present in test data point [True]
400 Text feature [accepted] present in test data point [True]
408 Text feature [018] present in test data point [True]
429 Text feature [19] present in test data point [True]
431 Text feature [better] present in test data point [True]
446 Text feature [acquire] present in test data point [True]
```

```
Out of the top  500  features  16 are present in query point
```

4.5 Random Forest Classifier
4.5.1. Hyper paramter tuning (With One hot Encoding)

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

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

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

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


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

        alpha = [100,200,500,1000,2000]
        max_depth = [5, 10]
        cv_log_error_array = []
        for i in alpha:
            for j in max_depth:
                print("for n_estimators =", i,"and max depth = ", j)
                clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
```

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

    '''fig, ax = plt.subplots()
    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_erro
    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', ma
    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 i
    predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation
    predict_y = sig_clf.predict_proba(test_x_onehotCoding)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.0878481048568867
for n_estimators = 100 and max depth =  10
Log Loss : 1.0859229689330498
for n_estimators = 200 and max depth =  5
Log Loss : 1.0761110656287374
for n_estimators = 200 and max depth =  10
Log Loss : 1.075561315216472
for n_estimators = 500 and max depth =  5
Log Loss : 1.0545388898412382
for n_estimators = 500 and max depth =  10
Log Loss : 1.061003654847057
for n_estimators = 1000 and max depth =  5
Log Loss : 1.0496849657598895
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0575168679753266
```

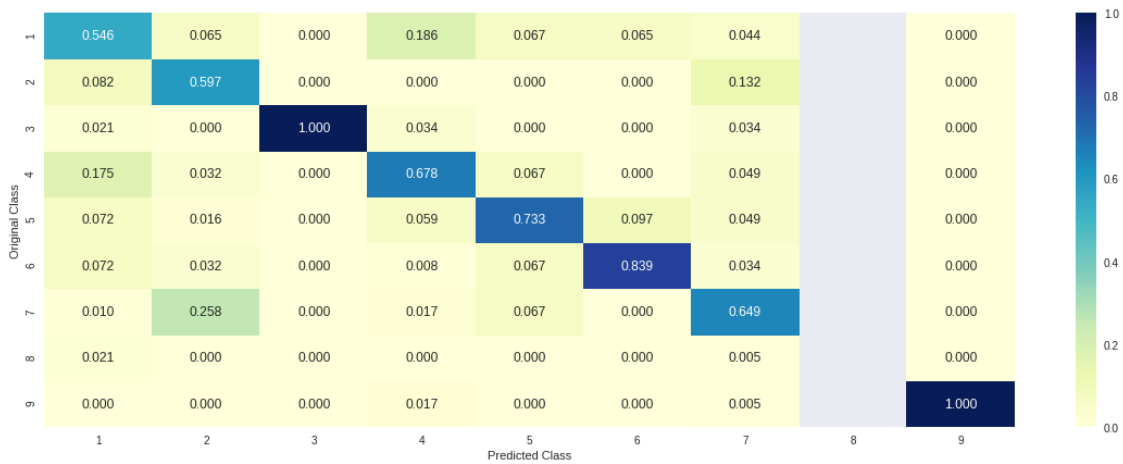
```
for n_estimators = 2000 and max depth =  5
Log Loss : 1.053235156485156
for n_estimators = 2000 and max depth =  10
Log Loss : 1.0562662926526651
For values of best estimator =  1000 The train log loss is: 0.5010319346377518
For values of best estimator =  1000 The cross validation log loss is: 1.0496849657598892
For values of best estimator =  1000 The test log loss is: 1.0850283535168812
```

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

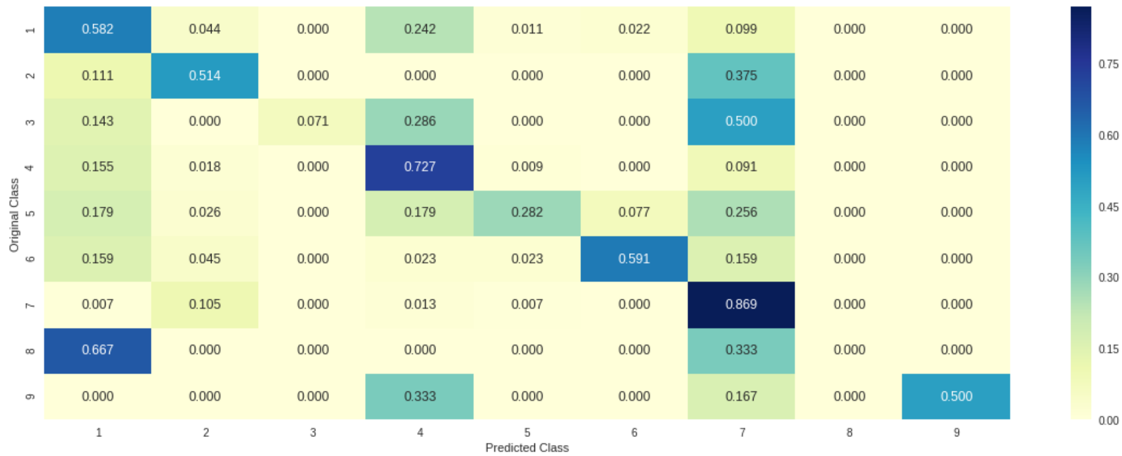
```
In [0]: # --------------------------------
        # default parameters
        # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
        # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
        # min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.0496849657598895
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```

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



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



4.5.3. Feature Importance

4.5.3.1. Incorrectly Classified point

In [0]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5481 0.003  0.013  0.0831 0.0686 0.2577 0.0079 0.0089 0.0097]
Actual Class : 5
--------------------------------------------------
33 Text feature [bars] present in test data point [True]
88 Text feature [agrees] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3.2. Correctly Classified point

In [0]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
```

```
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
        print("Actuall Class :", test_y[test_point_index])
        indices = np.argsort(-clf.feature_importances_)
        print("-"*50)
        get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_d
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0059 0.5592 0.01   0.0016 0.022  0.0301 0.3677 0.0026 0.0008]
Actuall Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

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

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

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

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


        # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
        # ----------------------------
        # 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
        #-------------------------------------
```

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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_erro
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 1.9127793983840824
for n_estimators = 10 and max depth =  3
Log Loss : 1.763485987753924
for n_estimators = 10 and max depth =  5
Log Loss : 1.4488155003302507
```

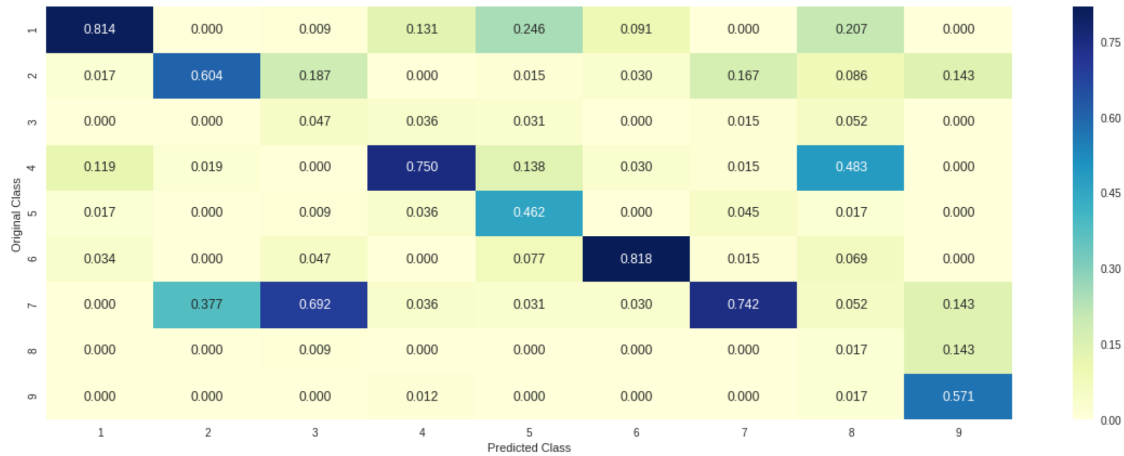
```
for n_estimators = 10 and max depth =  10
Log Loss : 1.8419722721053078
for n_estimators = 50 and max depth =  2
Log Loss : 1.9673705750403359
for n_estimators = 50 and max depth =  3
Log Loss : 1.8237682933574442
for n_estimators = 50 and max depth =  5
Log Loss : 1.4550214741781407
for n_estimators = 50 and max depth =  10
Log Loss : 1.5931056223316384
for n_estimators = 100 and max depth =  2
Log Loss : 1.7672483627659366
for n_estimators = 100 and max depth =  3
Log Loss : 1.5528446068054995
for n_estimators = 100 and max depth =  5
Log Loss : 1.4033635879804045
for n_estimators = 100 and max depth =  10
Log Loss : 1.637130499330871
for n_estimators = 200 and max depth =  2
Log Loss : 1.8718726737582434
for n_estimators = 200 and max depth =  3
Log Loss : 1.6332398556193024
for n_estimators = 200 and max depth =  5
Log Loss : 1.4559833997741711
for n_estimators = 200 and max depth =  10
Log Loss : 1.6273062778327712
for n_estimators = 500 and max depth =  2
Log Loss : 1.8819983259888995
for n_estimators = 500 and max depth =  3
Log Loss : 1.656446311689568
for n_estimators = 500 and max depth =  5
Log Loss : 1.5566220448130892
for n_estimators = 500 and max depth =  10
Log Loss : 1.6713707278386798
for n_estimators = 1000 and max depth =  2
Log Loss : 1.8815501766398273
for n_estimators = 1000 and max depth =  3
Log Loss : 1.7124729166623915
for n_estimators = 1000 and max depth =  5
Log Loss : 1.5698287813538716
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6687965685687636
For values of best alpha =  100 The train log loss is: 0.04242023486360206
For values of best alpha =  100 The cross validation log loss is: 1.403363587980406
For values of best alpha =  100 The test log loss is: 1.4705380097811172
```

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

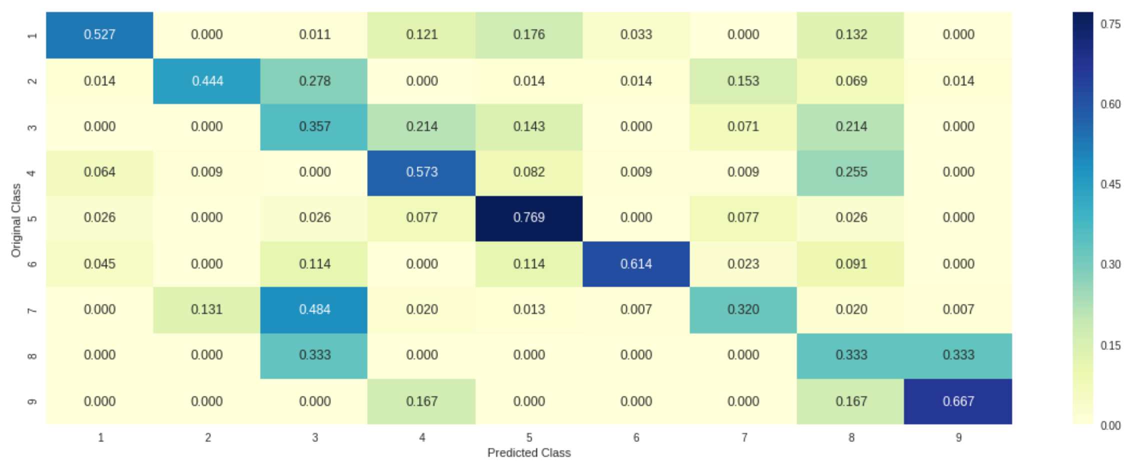
```
In [0]: # ------------------------------
        # default parameters
        # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
        # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
        # min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.4033635879804045
Number of mis-classified points : 0.5131578947368421
-------------------- Confusion matrix --------------------
```


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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
In [0]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', ma
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)


        test_point_index = 2
        no_feature = 27
        predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
        print("Predicted Class :", predicted_cls[0])
```

```python
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCo
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.feature_importances_)
        print("-"*50)
        for i in indices:
            if i<9:
                print("Gene is important feature")
            elif i<18:
                print("Variation is important feature")
            else:
                print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0093 0.0078 0.1009 0.8259 0.0157 0.0121 0.0054 0.0118 0.0112]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
```

```
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
```

4.5.5.2. Incorrectly Classified point

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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0285 0.2197 0.2781 0.023  0.1165 0.0552 0.1404 0.0918 0.0468]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
```

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

4.7 Stack the models
4.7.1 testing with hyper parameter tuning

In [0]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

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

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

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


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

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


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

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

```
            sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifie
            sclf.fit(train_x_onehotCoding, train_y)
            print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_lo
            log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
            if best_alpha > log_error:
                best_alpha = log_error
```

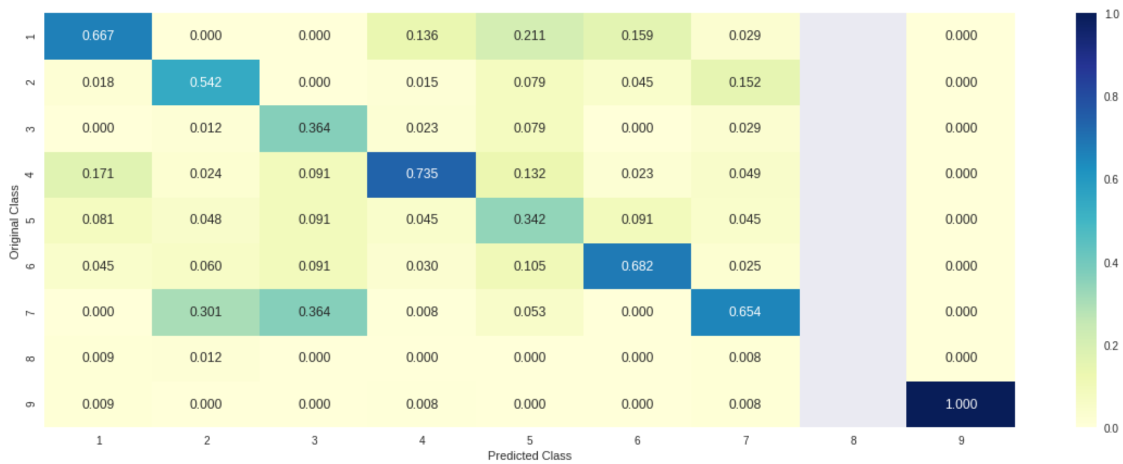
```
Logistic Regression :  Log Loss: 0.96
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.22
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.478
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.079
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.154
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.362
```

4.7.2 testing the model with the best hyper parameters

In [0]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.6372082703349873
Log loss (CV) on the stacking classifier : 1.0788833027194469
Log loss (test) on the stacking classifier : 1.116622642202851
Number of missclassified point : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

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

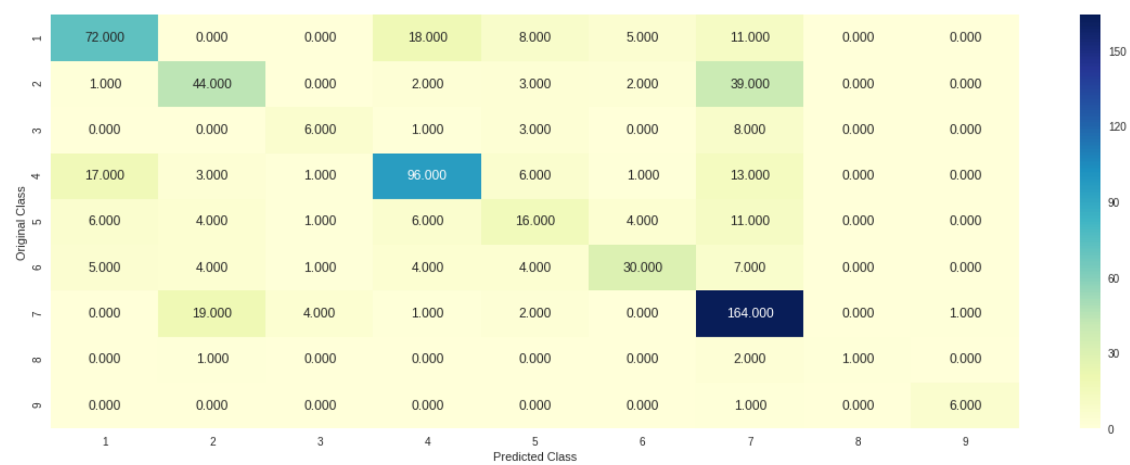


-------------------- Recall matrix (Row sum=1) --------------------

4.7.3 Maximum Voting classifier

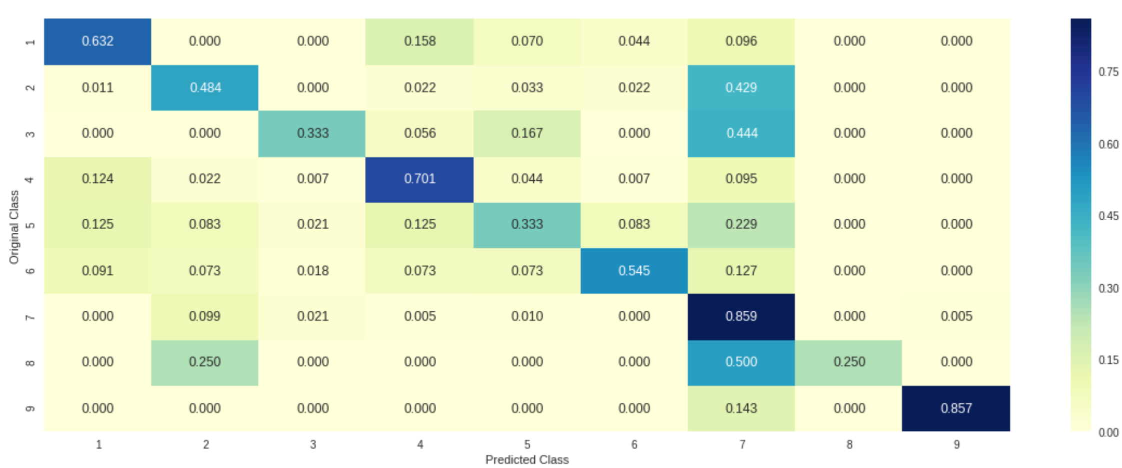
```
In [0]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifie
        from sklearn.ensemble import VotingClassifier
        vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3
        vclf.fit(train_x_onehotCoding, train_y)
        print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba
        print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_
        print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(t
        print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCo
        plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.869377753269717
Log loss (CV) on the VotingClassifier : 1.1385756568426066
Log loss (test) on the VotingClassifier : 1.172654824350643
Number of missclassified point : 0.3458646616541353
-------------------- Confusion matrix --------------------
```



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

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



5. Assignments

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

```
In [15]: # CONCLUSION

         # Please compare all your models using Prettytable library
         from prettytable import PrettyTable

         x = PrettyTable()

         x.field_names = ["SERIAL NO","CONDITON","MODEL","TEXT_LOG_LOSS","TEST_LOG_LOSS" ,"CV_LO

         x.add_row([1,'TFIDF','NAIVE_BAYES', 0.83, 1.30, 1.288,  0.44 ])
         x.add_row([2,'TFIDF','KNN', 0.62,1.083,1.083,.36 ])
         x.add_row([3,'TFIDF','LOGISTIC(BALANCED)', 0.566,1.22,1.16,.39])
         x.add_row([4,'TFIDF','LOGISTIC(UNBALANCED)', .82,1.23,1.21,.38 ])
         x.add_row([5,'TFIDF','RANDOMFOREST',.072,1.36,1.44,.43])
         x.add_row([6,'TFIDF','STACK',.61,1.22,1.20,.39])
         x.add_row([7,'TFIDF','LINEAR',.63,1.22,1.21,.38 ])
         x.add_row([8,'TFIDF','MAX_VOTE',.84,1.20,1.209,.38 ])



         x.add_row([10,'TFIDF_max_features=1000','NAIVE_BAYES',.51,1.22,1.21,.39 ])
         x.add_row([11,'TFIDF_max_features=1000','KNN', .62,1.08,1.08,.36 ])
         x.add_row([12,'TFIDF_max_features=1000','LOGISTIC(BALANCED)', 0.43, 1.07, 1.04,  .39 ])
         x.add_row([13,'TFIDF_max_features=1000','LOGISTIC(UNBALANCED)', .42,1.12,1.07,.35 ])
         x.add_row([14,'TFIDF_max_features=1000','STACK',.53,1.20,1.19,.38])
         x.add_row([15,'TFIDF_max_features=1000','LINEAR',.47,1.08,1.09,.36 ])
         x.add_row([16,'TFIDF_max_features=1000','MAX_VOTE',.82,1.21,1.21,.38 ])

         x.add_row([17,'TFIDF_UNIGRAM_BIGRAM','LOGISTIC(BALANCED)', .70,1.28,1.29,.41])
         x.add_row([18,'TFIDF_UNIGRAM_BIGRAM','LOGISTIC(UNBALANCED)', .70,1.28,1.27,.42 ])
         x

         x.add_row([19,'TFIDF_max_features=10000','NAIVE_BAYES',.78,1.22,1.25,.36 ])
         x.add_row([20,'TFIDF_max_features=10000','KNN', .67,.96,1.04,.31])
         x.add_row([21,'TFIDF_max_features=10000','LOGISTIC(BALANCED)', .54,.96,1.01,.32 ])
         x.add_row([22,'TFIDF_max_features=10000','LOGISTIC(UNBALANCED)',.54,.97,1.02,.31 ])
         x.add_row([24,'TFIDF_max_features=10000','RANDOMFOREST',.52,1.00,1.07,.33])

         x.add_row([25,'TFIDF_max_features=10000','STACK',.63,1.07,1.11,.36])
         x.add_row([26,'TFIDF_max_features=10000','LINEAR',.52,1.00,1.07,.33 ])
         x.add_row([27,'TFIDF_max_features=10000','MAX_VOTE',.86,1.13,1.17,.34])


         print(x)

+-----------+-------------------------+---------------------+---------------+---------------+-
| SERIAL NO |         CONDITON        |        MODEL        | TEXT_LOG_LOSS | TEST_LOG_LOSS |
+-----------+-------------------------+---------------------+---------------+---------------+-
```

1	TFIDF	NAIVE_BAYES	0.83	1.3
2	TFIDF	KNN	0.62	1.083
3	TFIDF	LOGISTIC(BALANCED)	0.566	1.22
4	TFIDF	LOGISTIC(UNBALANCED)	0.82	1.23
5	TFIDF	RANDOMFOREST	0.072	1.36
6	TFIDF	STACK	0.61	1.22
7	TFIDF	LINEAR	0.63	1.22
8	TFIDF	MAX_VOTE	0.84	1.2
10	TFIDF_max_features=1000	NAIVE_BAYES	0.51	1.22
11	TFIDF_max_features=1000	KNN	0.62	1.08
12	TFIDF_max_features=1000	LOGISTIC(BALANCED)	0.43	1.07
13	TFIDF_max_features=1000	LOGISTIC(UNBALANCED)	0.42	1.12
14	TFIDF_max_features=1000	STACK	0.53	1.2
15	TFIDF_max_features=1000	LINEAR	0.47	1.08
16	TFIDF_max_features=1000	MAX_VOTE	0.82	1.21
17	TFIDF_UNIGRAM_BIGRAM	LOGISTIC(BALANCED)	0.7	1.28
18	TFIDF_UNIGRAM_BIGRAM	LOGISTIC(UNBALANCED)	0.7	1.28
19	TFIDF_max_features=10000	NAIVE_BAYES	0.78	1.22
20	TFIDF_max_features=10000	KNN	0.67	0.96
21	TFIDF_max_features=10000	LOGISTIC(BALANCED)	0.54	0.96
22	TFIDF_max_features=10000	LOGISTIC(UNBALANCED)	0.54	0.97
24	TFIDF_max_features=10000	RANDOMFOREST	0.52	1.0
25	TFIDF_max_features=10000	STACK	0.63	1.07
26	TFIDF_max_features=10000	LINEAR	0.52	1.0
27	TFIDF_max_features=10000	MAX_VOTE	0.86	1.13

1 STEPS FOLLOWED TO ACHEIVE 69% ACCURACY WITH LOG LOSS =0.96

1) The goal was to acheive a logloss less than 1.The acheive log-loss is 0.96
2) We are provided with the text, the description of the personalized cancer diagnosis
3) preprocessing of the text file included following steps:
   a) Removing all the english,stopwords,custom words including 'mutation','cell'which are the most occuring words in all the class lebels
   b) I have removed 'mutation' and 'cell' considering that the description of the respective class labels should consist of distinct words for better training of the model c)Used snowball stemmer for better version of text
4) Feature Enginnearing inchulded following steps:

   a)Used the combination of gene,variation,text to form a new column in the dataframe b)considered the length of the text as a new column in the dataframe
5) stacked all the column to predict the label

6) Used tfidfVectorizer on the complete model which improved the accuray by approx 5% in some models
7) used tfidfVectorizer with max_features=1000 on all the model where some models like naive bayes,logistic without balancing,SVMlinear has worked comparitively better but other model didnot show any improvement
8) Used unigram and bigram on tfidf Used but the increased the count of misclassfied point on logistic regression as stated in the observation table
9) Tried to play with tfidfVectorizer with max_features=10000 considering that due to less max_features that is 1000 we might miss important features
10) The result improved very much and KNN and logistic regression show the best result with 0.96 and .97 and accuracy =69, misclassified points =0.31
11) WE could try more feature enginnering techniques like number of letter of the text and playing and experimenting with them