CancerDiagnosis-Assignment-2

July 24, 2018

1 Importing the libraries

In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier
```

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This
  "This module will be removed in 0.20.", DeprecationWarning)
```

3.1. Reading Data

```
In [6]: data = pd.read_csv('training_variants')
        print('Number of data points : ', data.shape[0])
        print('Number of features : ', data.shape[1])
        print('Features : ', data.columns.values)
        data.head()
```

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

```
Out[6]:    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
```

```
In [7]: # 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[7]:    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...
```

```
In [8]: import nltk
        nltk.download('stopwords')
        import re
```

```
[nltk_data] Downloading package stopwords to /content/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [0]:
```python
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [10]:
```python
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[10]:
```
   ID    Gene             Variation  Class  \
0   0  FAM58A  Truncating Mutations      1
1   1     CBL                 W802*      2
2   2     CBL                 Q249E      2
3   3     CBL                 N454D      3
4   4     CBL                 L399V      4

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

3.1.4. Test, Train and Cross Validation Split

In [0]:
```python
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 varaib
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, t
# split the train data into train and cross validation by maintaining same distributio
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, t
```

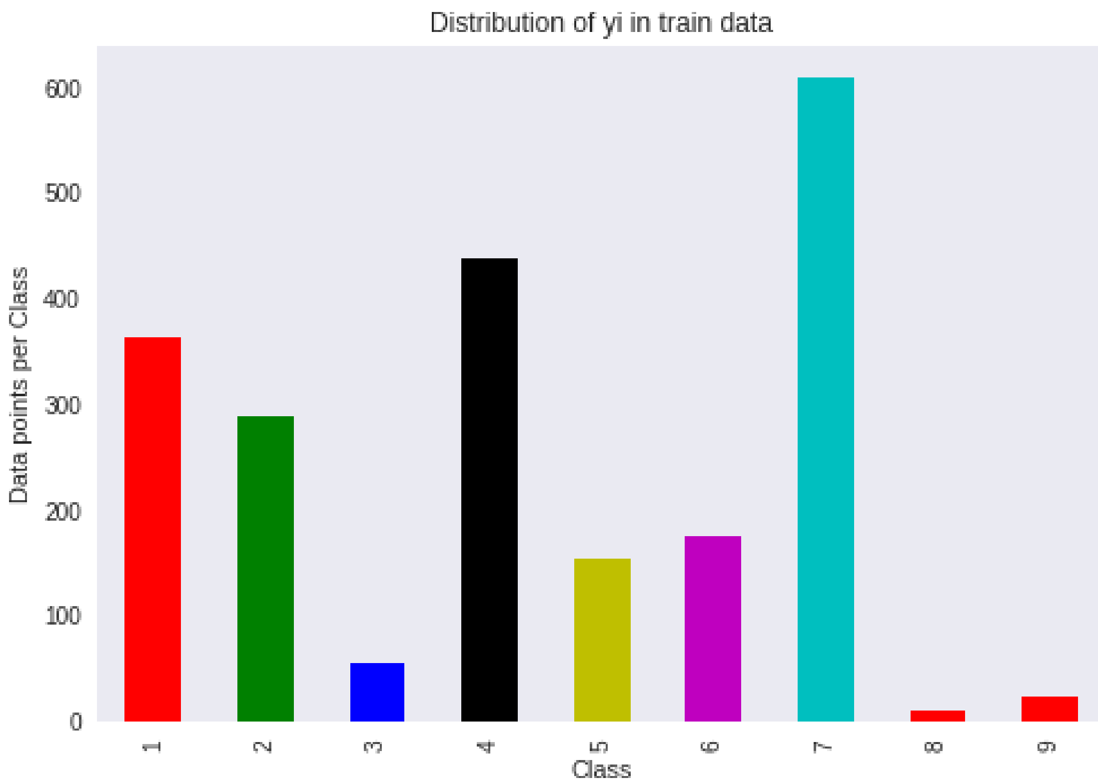
```
In [12]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

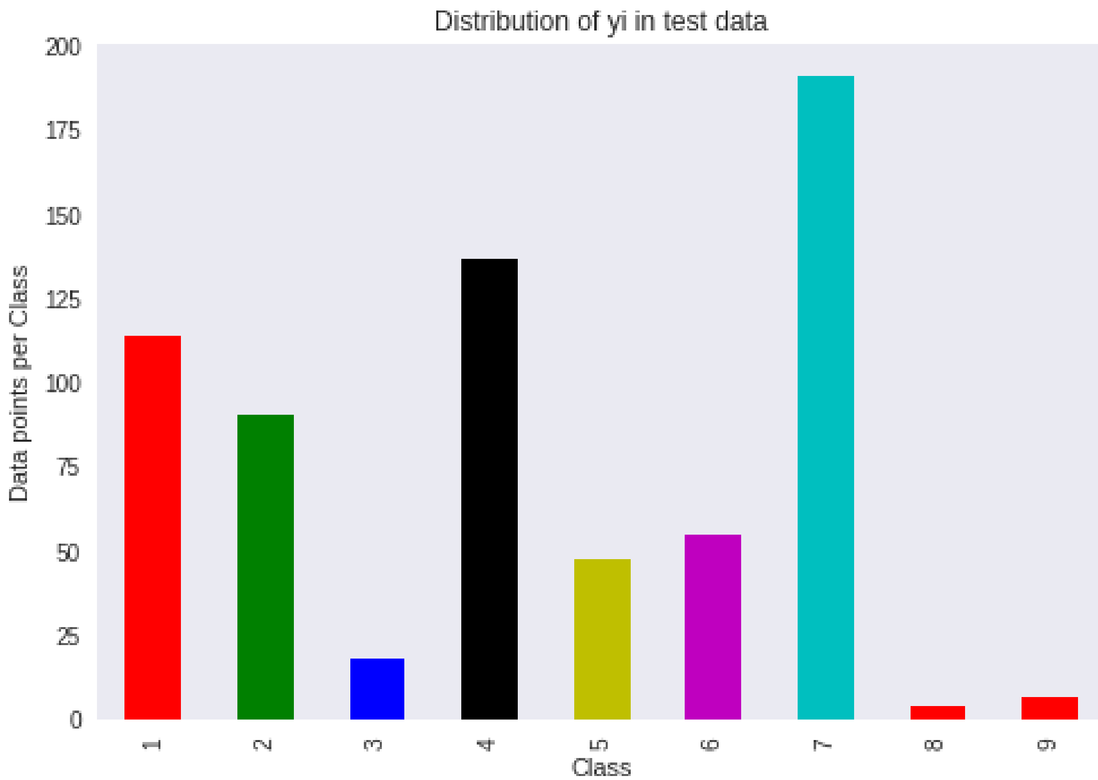
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [13]: # it returns a dict, keys as class labels and values as the number of data points in
         train_class_distribution = train_df['Class'].value_counts().sortlevel()
         test_class_distribution = test_df['Class'].value_counts().sortlevel()
         cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

         my_colors =  ['r', 'g', 'b', 'k', 'y', 'm', 'c']
         train_class_distribution.plot(kind='bar', color=my_colors)
         plt.xlabel('Class')
         plt.ylabel('Data points per Class')
         plt.title('Distribution of yi in train data')
         plt.grid()
         plt.show()

         sorted_yi = np.argsort(-train_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':',train_class_distribution.values[i


         print('-'*80)
         my_colors =  ['r', 'g', 'b', 'k', 'y', 'm', 'c']
         test_class_distribution.plot(kind='bar', color=my_colors)
         plt.xlabel('Class')
         plt.ylabel('Data points per Class')
         plt.title('Distribution of yi in test data')
         plt.grid()
         plt.show()

         sorted_yi = np.argsort(-test_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':',test_class_distribution.values[i]

         print('-'*80)
         my_colors =  ['r', 'g', 'b', 'k', 'y', 'm', 'c']
```

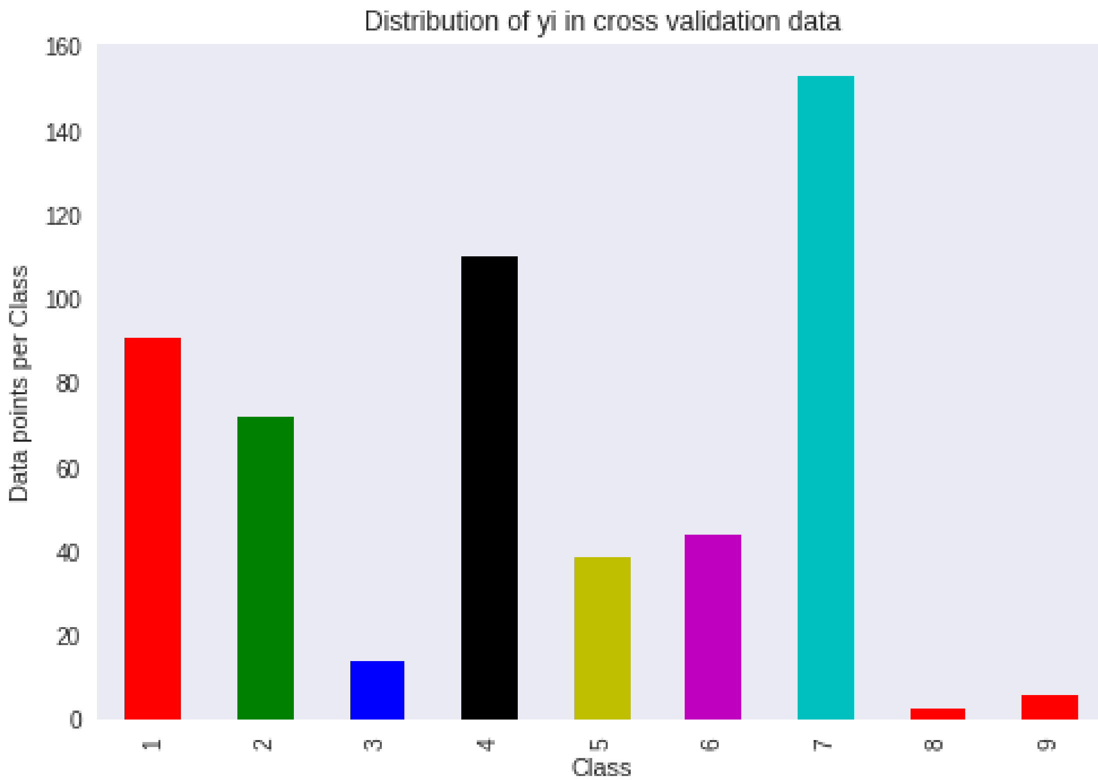
```
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],
```




```
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 %)
```

2 Tf-idf Vectorization

In [0]:
```
# Gene and Variation feature
gene_vectorizer = TfidfVectorizer()
train_gene_feature_tfidfCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tfidfCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tfidfCoding = gene_vectorizer.transform(cv_df['Gene'])
```

```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_tfidfCoding = variation_vectorizer.fit_transform(train_df['Vari
test_variation_feature_tfidfCoding = variation_vectorizer.transform(test_df['Variation
cv_variation_feature_tfidfCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [16]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_tfidfCoding = text_vectorizer.fit_transform(train_df['TEXT'].values
# we use the same vectorizer that was trained on train data
test_text_feature_tfidfCoding = text_vectorizer.transform(test_df['TEXT'].values.astyp
# we use the same vectorizer that was trained on train data
cv_text_feature_tfidfCoding = text_vectorizer.transform(cv_df['TEXT'].values.astype('U
# 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*num
train_text_fea_counts = train_text_feature_tfidfCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 55047
```

2.0.1 Extracting top 1000 features

In [0]:
```
tfidf_transp = train_text_feature_tfidfCoding.T
feature_max = {}
for i in range(len(train_text_features)):
    feature_max[train_text_features[i]] = tfidf_transp[i].max()

top1000_idx = np.argsort(-np.array(list(feature_max.values())))[0:1000]


train_text_feature_tfidfCoding = train_text_feature_tfidfCoding[:,top1000_idx]
test_text_feature_tfidfCoding = test_text_feature_tfidfCoding[:,top1000_idx]
cv_text_feature_tfidfCoding = cv_text_feature_tfidfCoding[:,top1000_idx]
```

2.0.2 Stacking Features

In [0]:
```
train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_fea
test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_featu
cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding,cv_variation_feature_tfi

train_x_tfidfCoding = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfCodi
train_y = np.array(list(train_df['Class']))
```

```
test_x_tfidfCoding = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCoding)
test_y = np.array(list(test_df['Class']))

cv_x_tfidfCoding = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding)).tocs
cv_y = np.array(list(cv_df['Class']))
```

3 Machine Learning Models

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 predi

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

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

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

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

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

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

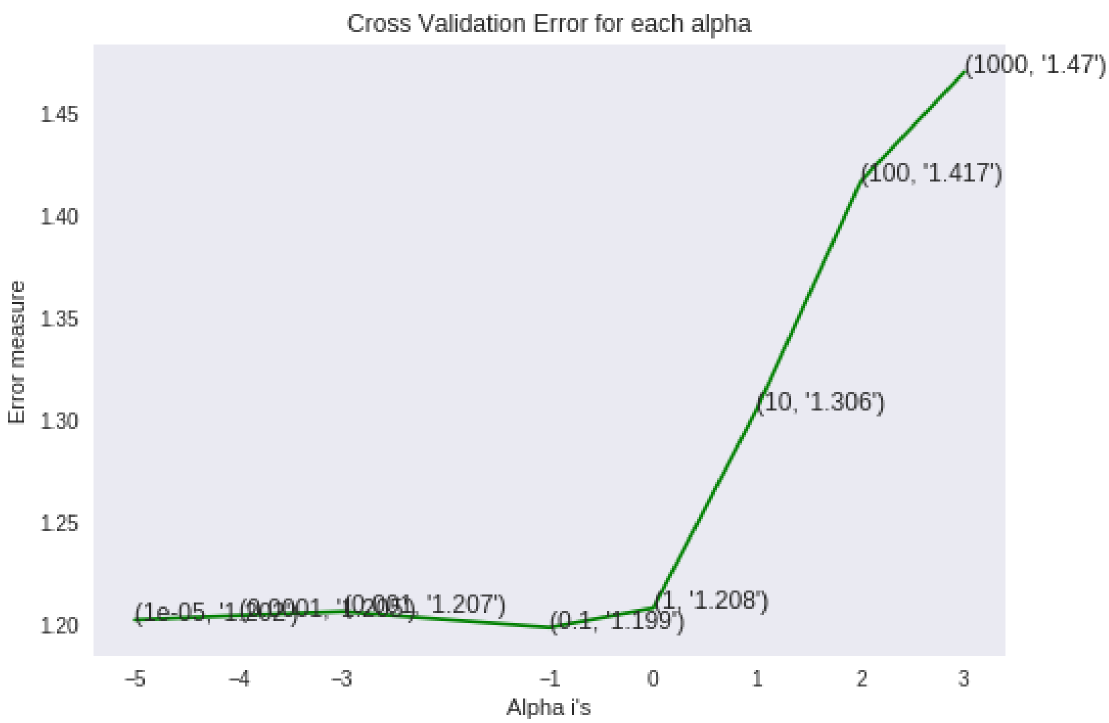
```
In [0]: def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
            clf.fit(train_x, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x, train_y)
            pred_y = sig_clf.predict(test_x)

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

3.0.1 Naive Bayes

```
In [0]: 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_tfidfCoding, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x_tfidfCoding, train_y)
            sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
            cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
            # to avoid rounding error while multiplying probabilites we use log-probability es
            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_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-05
Log Loss : 1.2024943539431565
for alpha = 0.0001
Log Loss : 1.2046601587119516
for alpha = 0.001
Log Loss : 1.2065834023003155
for alpha = 0.1
Log Loss : 1.1987075294046474
for alpha = 1
Log Loss : 1.208311582092767
for alpha = 10
Log Loss : 1.305596166324385
for alpha = 100
Log Loss : 1.4166775347789524
for alpha = 1000
Log Loss : 1.4700461514977432
```

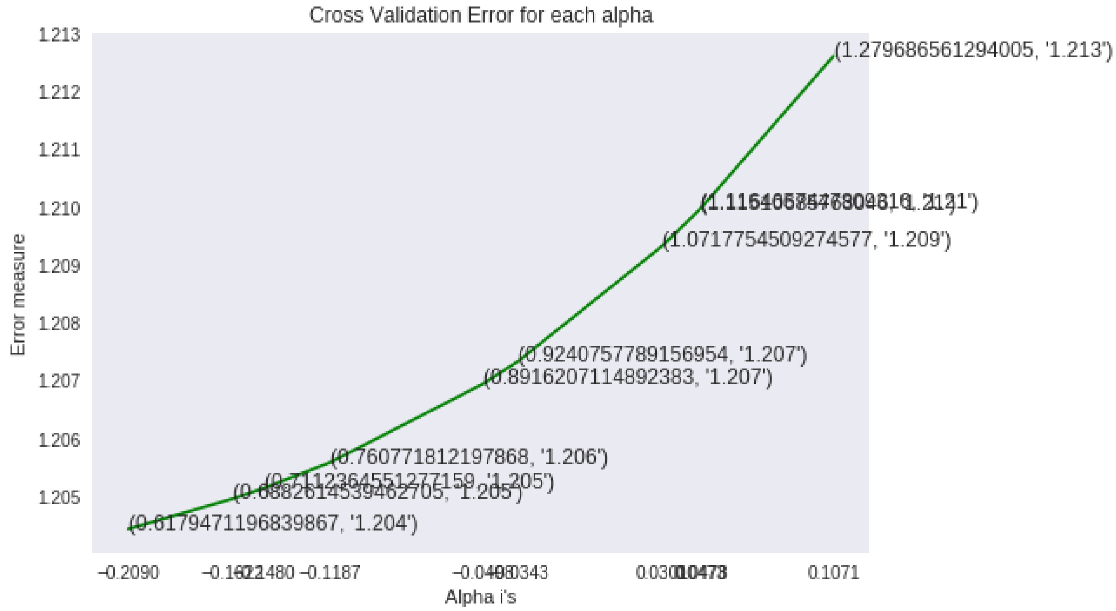



```
For values of best alpha =  0.1 The train log loss is: 0.5713616818640016
For values of best alpha =  0.1 The cross validation log loss is: 1.1987075294046474
For values of best alpha =  0.1 The test log loss is: 1.2210674713328047
```

```
In [0]: alpha = np.random.uniform(0.5,1.5,10)
        alpha.sort()
        cv_log_error_array = []
        for i in alpha:
            print("for alpha =", i)
            clf = MultinomialNB(alpha=i)
            clf.fit(train_x_tfidfCoding, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x_tfidfCoding, train_y)
            sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
            cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
            # to avoid rounding error while multiplying probabilites we use log-probability es
            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()
```

```python
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_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

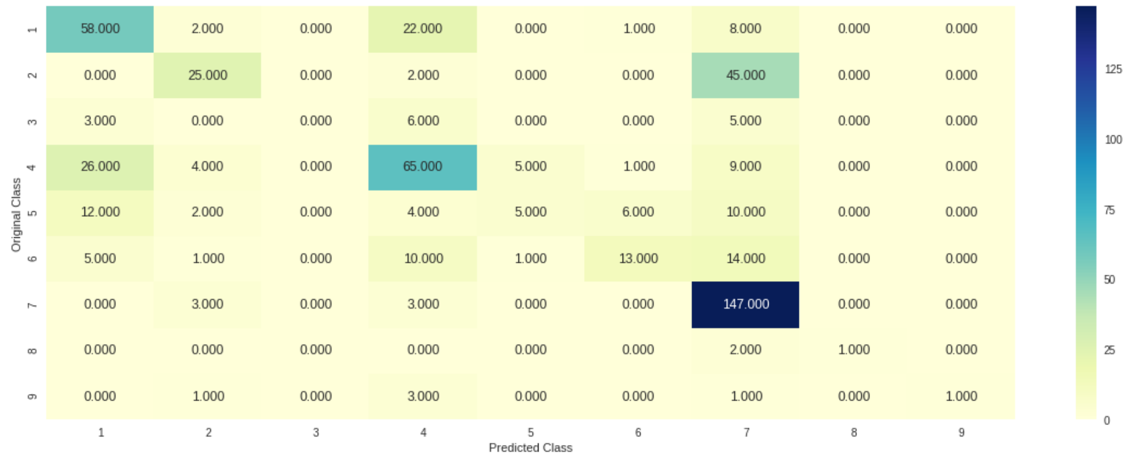
```
for alpha = 0.6179471196839867
Log Loss : 1.204437387820135
for alpha = 0.6882614539462705
Log Loss : 1.2049570608562095
for alpha = 0.7112364551277159
Log Loss : 1.2051479586463165
for alpha = 0.760771812197868
Log Loss : 1.2055819077882497
for alpha = 0.8916207114892383
Log Loss : 1.2069486331774397
for alpha = 0.9240757789156954
Log Loss : 1.2073354784124894
for alpha = 1.0717754509274577
Log Loss : 1.2093216729938234
for alpha = 1.11510685763046
Log Loss : 1.209968319487109
for alpha = 1.1164657447809316
Log Loss : 1.2099890055120186
for alpha = 1.279686561294005
Log Loss : 1.2126148715414744
```



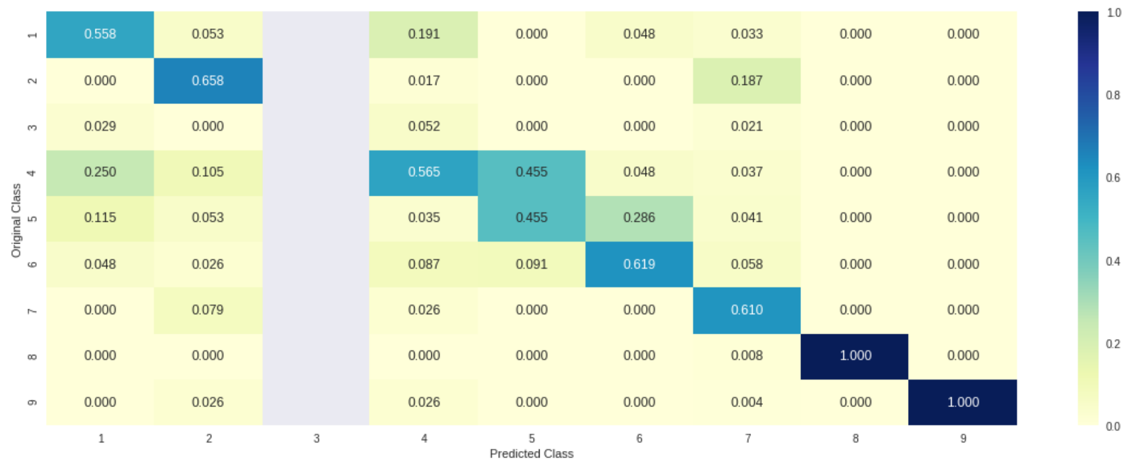

```
For values of best alpha =  0.6179471196839867 The train log loss is: 0.7630199856270696
For values of best alpha =  0.6179471196839867 The cross validation log loss is: 1.20443738782
For values of best alpha =  0.6179471196839867 The test log loss is: 1.1996418469843348
```

```
In [0]: ##error and confusioon matrix
        final_results = []
        clf = MultinomialNB(alpha=1.3216328879483359)
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
        # to avoid rounding error while multiplying probabilites we use log-probability estima
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
        print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfidf
        plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidfCoding.toarray()))
```

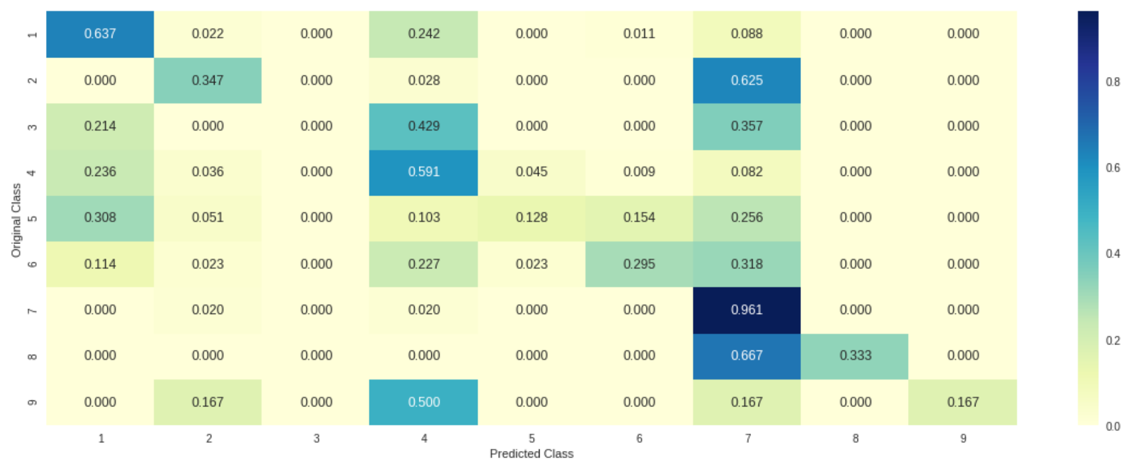
```
Log Loss : 1.2133222540229425
Number of missclassified point : 0.40789473684210525
-------------------- Confusion matrix --------------------
```



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



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



Feature Imortance

```
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['TEXT'])

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

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

```
                    word_present += 1
                    print(i, "variation feature [{}] present in test data point [{}]".forma
            else:
                temp1 = list(np.array(text_vec.get_feature_names())[top1000_idx])
                word = temp1[v-(fea1_len+fea2_len)]
                yes_no = True if word in text.split() else False
                if yes_no:
                    word_present += 1
                    print(i, "Text feature [{}] present in test data point [{}]".format(wo

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

In [0]:
```
test_point_index = 75
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
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 : 1
Predicted Class Probabilities: [[0.3346 0.055  0.0228 0.2836 0.1991 0.0765 0.0202 0.0041 0.004
Actual Class : 4
--------------------------------------------------
0 Text feature [the] present in test data point [True]
1 Text feature [of] present in test data point [True]
2 Text feature [and] present in test data point [True]
3 Text feature [in] present in test data point [True]
6 Text feature [to] present in test data point [True]
9 Text feature [p53] present in test data point [True]
13 Text feature [variants] present in test data point [True]
15 Text feature [cells] present in test data point [True]
18 Text feature [et] present in test data point [True]
23 Text feature [cell] present in test data point [True]
25 Text feature [protein] present in test data point [True]
41 Text feature [patients] present in test data point [True]
68 Text feature [families] present in test data point [True]
71 Text feature [ovarian] present in test data point [True]
85 Text feature [deleterious] present in test data point [True]
Out of the top  100  features  15 are present in query point
```

In [0]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
        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']
```

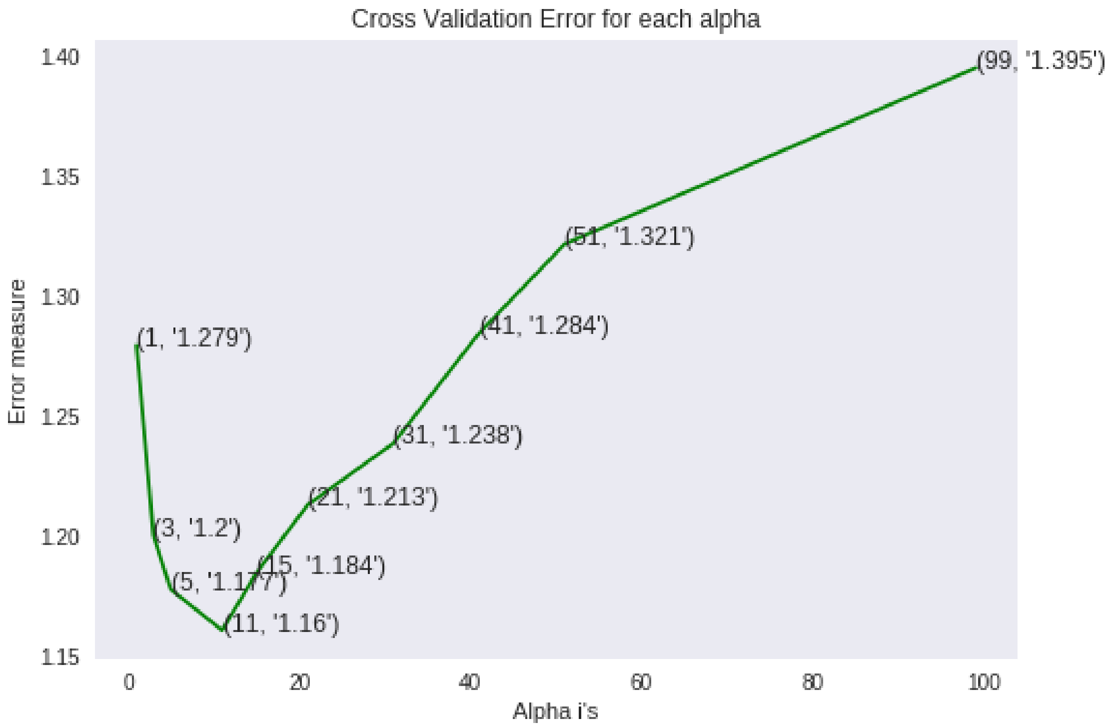
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0362 0.0683 0.0249 0.047  0.0442 0.0426 0.7256 0.0048 0.006
Actual Class : 7
--------------------------------------------------
0 Text feature [the] present in test data point [True]
1 Text feature [of] present in test data point [True]
2 Text feature [and] present in test data point [True]
3 Text feature [in] present in test data point [True]
4 Text feature [to] present in test data point [True]
6 Text feature [cells] present in test data point [True]
17 Text feature [cell] present in test data point [True]
21 Text feature [et] present in test data point [True]
41 Text feature [kit] present in test data point [True]
47 Text feature [resistance] present in test data point [True]
52 Text feature [protein] present in test data point [True]
79 Text feature [crizotinib] present in test data point [True]
87 Text feature [melanoma] present in test data point [True]
Out of the top  100  features  13 are present in query point
```

4 K nearest neighbour

```
In [0]: alpha = [1, 3, 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_tfidfCoding, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x_tfidfCoding, train_y)
            sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
            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 es
            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_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss 
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

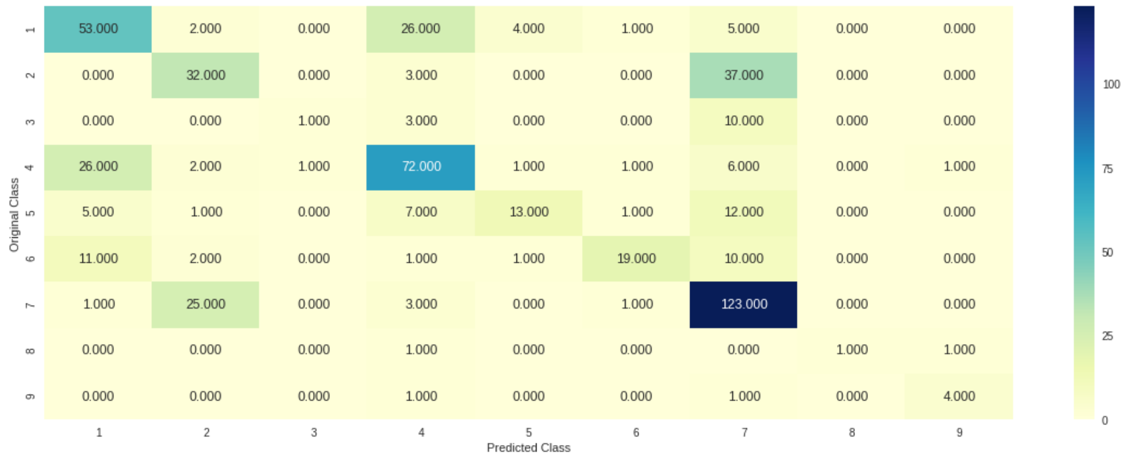
```
for alpha = 1
Log Loss : 1.2791189483038246
for alpha = 3
Log Loss : 1.1996782482489654
for alpha = 5
Log Loss : 1.1774296048276554
for alpha = 11
Log Loss : 1.1603520012368613
for alpha = 15
Log Loss : 1.1842230738723414
for alpha = 21
Log Loss : 1.2126623104898118
for alpha = 31
Log Loss : 1.2382648463914498
for alpha = 41
Log Loss : 1.2841301431035497
for alpha = 51
Log Loss : 1.3213059157557268
for alpha = 99
Log Loss : 1.3946731900283258
```



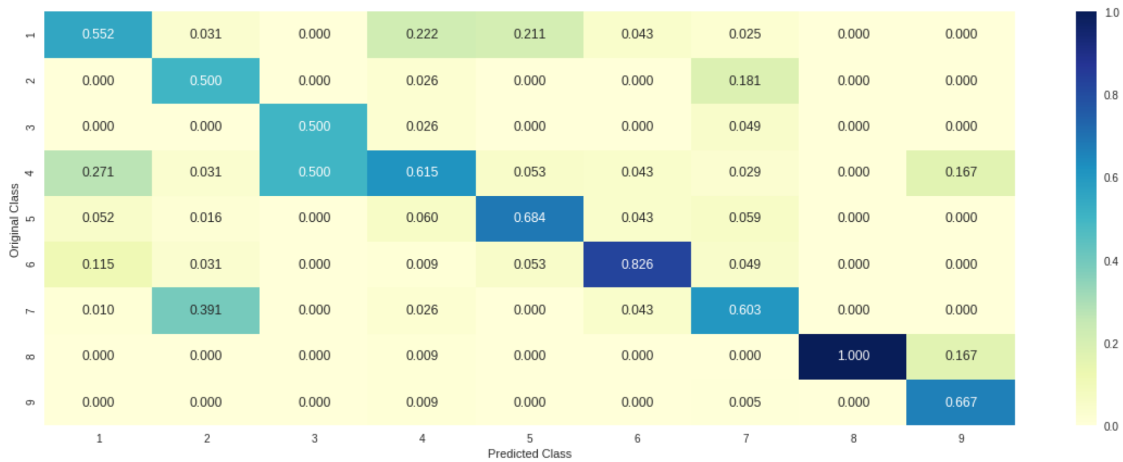

```
For values of best alpha =  11 The train log loss is: 0.9861917142457807
For values of best alpha =  11 The cross validation log loss is: 1.1603520012368613
For values of best alpha =  11 The test log loss is: 1.2165359533126943
```

```
In [23]: #testing
         clf = KNeighborsClassifier(n_neighbors=11,algorithm='brute')
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y
```

```
Log loss : 1.1449872906397758
Number of mis-classified points : 0.40225563909774437
-------------------- Confusion matrix --------------------
```



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



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

4.0.1 Feature importance

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

        test_point_index = 45
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Actual Class :", test_y[test_point_index])
        neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
        print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to class
        print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

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

        test_point_index = 95
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Actual Class :", test_y[test_point_index])
        neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
```

```
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to class
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

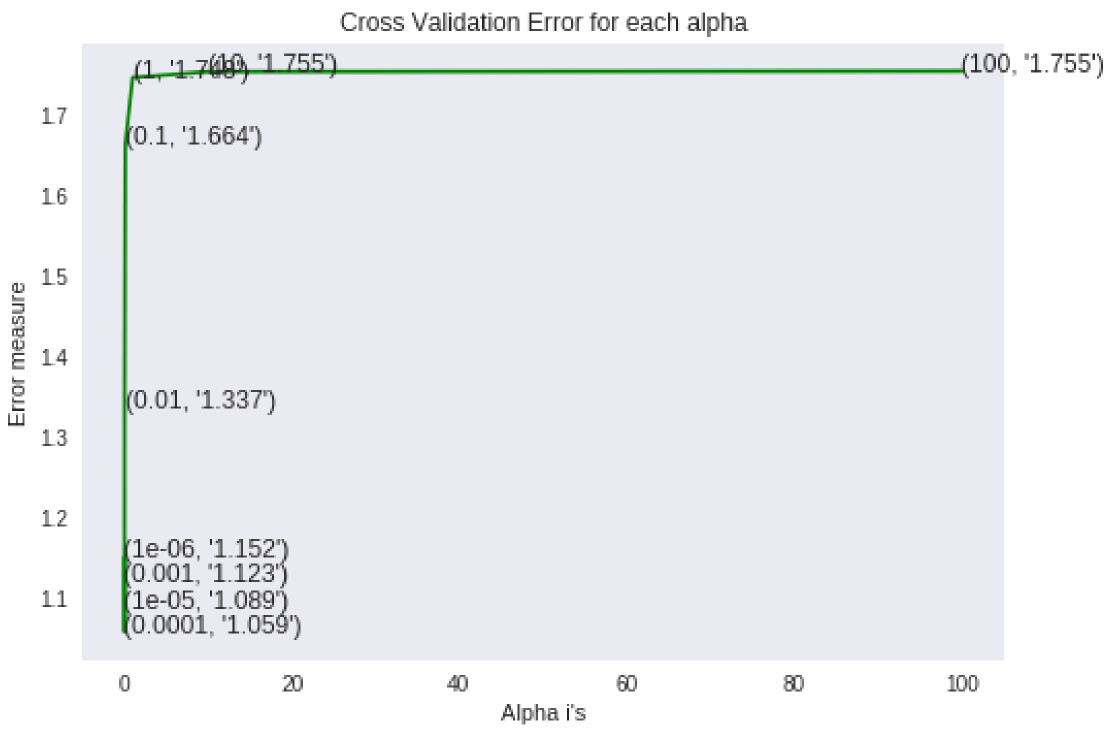
5 Logistic Regression

5.0.1 With Class Balancing

In [0]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ran
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
    # to avoid rounding error while multiplying probabilites we use log-probability es
    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', los
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

```python
    predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 1e-06
Log Loss : 1.1522409047638515
for alpha = 1e-05
Log Loss : 1.089104128241553
for alpha = 0.0001
Log Loss : 1.0587567832126128
for alpha = 0.001
Log Loss : 1.1229143574211924
for alpha = 0.01
Log Loss : 1.3374477194479193
for alpha = 0.1
Log Loss : 1.664089392065227
for alpha = 1
Log Loss : 1.7475557413448564
for alpha = 10
Log Loss : 1.7546836216595703
for alpha = 100
Log Loss : 1.7554240462737334
```



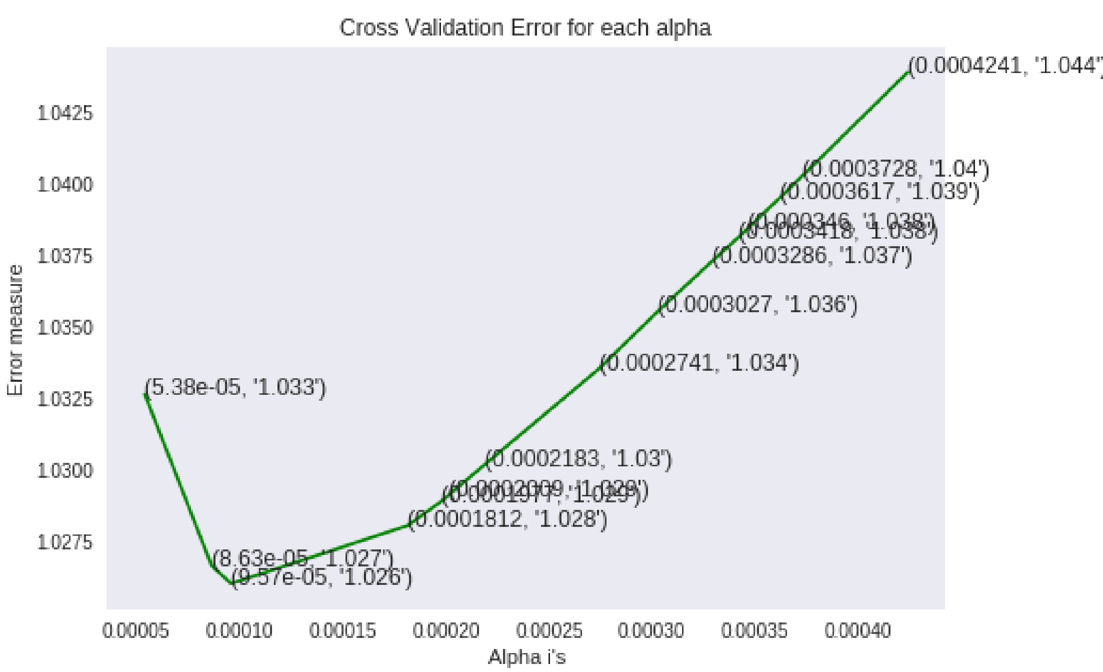

```
For values of best alpha =  0.0001 The train log loss is: 0.4441015530403704
For values of best alpha =  0.0001 The cross validation log loss is: 1.0587567832126128
```

```
For values of best alpha =  0.0001 The test log loss is: 1.080897787920164
```

```
In [24]: alpha = np.random.uniform(0.00005,0.0005,15)
         alpha = np.round(alpha,7)
         alpha.sort()
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
             clf.fit(train_x_tfidfCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_tfidfCoding, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
             # to avoid rounding error while multiplying probabilites we use log-probability e
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5.38e-05
Log Loss : 1.0326360321470949
for alpha = 8.63e-05
Log Loss : 1.026623835565087
for alpha = 9.57e-05
Log Loss : 1.026003811355244
```

```
for alpha = 0.0001812
Log Loss : 1.0280294372976273
for alpha = 0.0001977
Log Loss : 1.0288834167641248
for alpha = 0.0002009
Log Loss : 1.0290783362634195
for alpha = 0.0002183
Log Loss : 1.030140530257667
for alpha = 0.0002741
Log Loss : 1.0335155245077763
for alpha = 0.0003027
Log Loss : 1.0355276787534429
for alpha = 0.0003286
Log Loss : 1.0372547642505963
for alpha = 0.0003418
Log Loss : 1.0381245226666742
for alpha = 0.000346
Log Loss : 1.0384087204531722
for alpha = 0.0003617
Log Loss : 1.0394919933170037
for alpha = 0.0003728
Log Loss : 1.0402698925042402
for alpha = 0.0004241
Log Loss : 1.0439114898141926
```

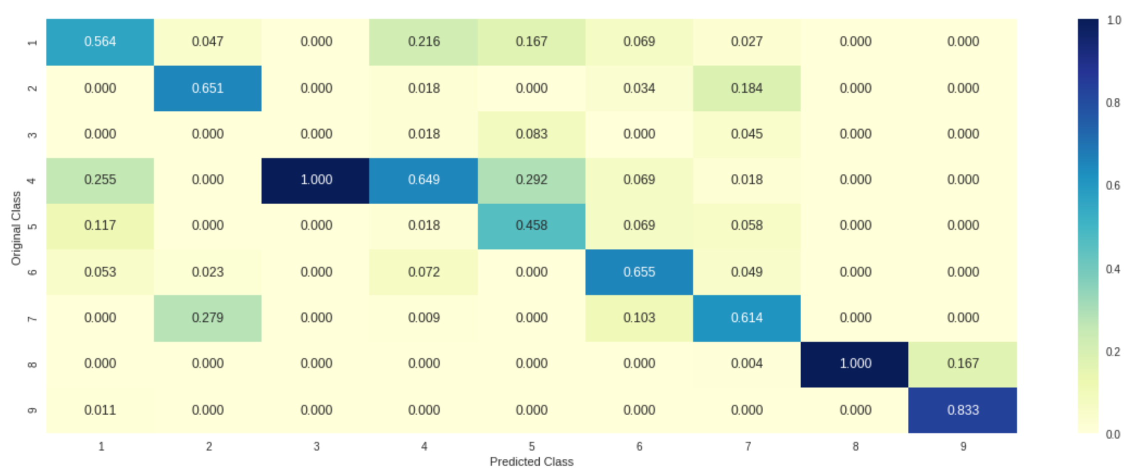


```
For values of best alpha =  9.57e-05 The train log loss is: 0.4516881446943588
For values of best alpha =  9.57e-05 The cross validation log loss is: 1.026003811355244
For values of best alpha =  9.57e-05 The test log loss is: 1.0636988898227784
```

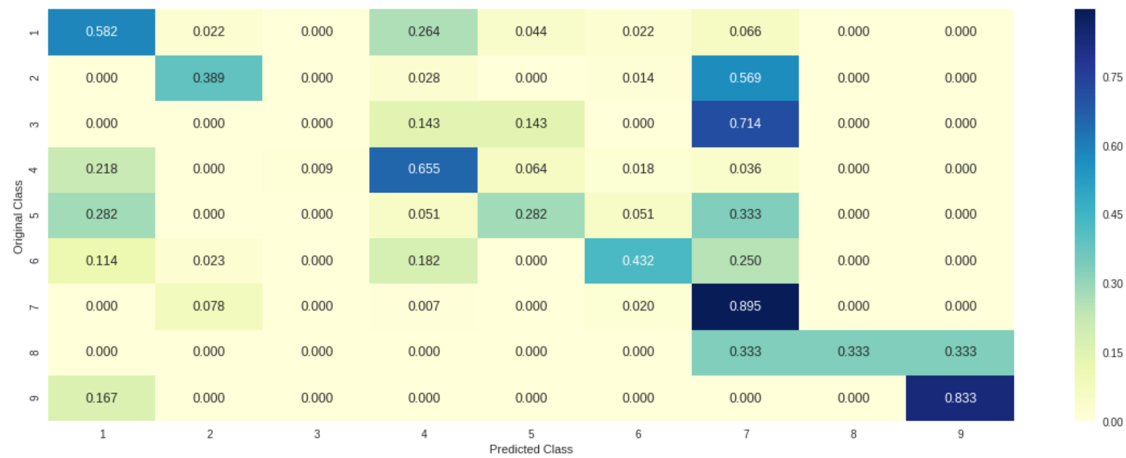
In [25]:
```
#testing
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y
```

```
Log loss : 1.026003811355244
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```


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



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



5.0.2 Feature Importance

```
In [0]: clf = SGDClassifier(class_weight='balanced', alpha=0.00013, penalty='l2', loss='log', r
        clf.fit(train_x_tfidfCoding,train_y)
        test_point_index = 1
        no_feature = 500
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
        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 : 7
Predicted Class Probabilities: [[0.0367 0.021  0.0159 0.0305 0.0971 0.0243 0.6558 0.0064 0.1123
Actual Class : 7
--------------------------------------------------
10 Text feature [cells] present in test data point [True]
274 Text feature [kit] present in test data point [True]
285 Text feature [crizotinib] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

```
In [0]: clf = SGDClassifier(class_weight='balanced', alpha=0.00013, penalty='l2', loss='log', r
        clf.fit(train_x_tfidfCoding,train_y)
        test_point_index = 15
        no_feature = 500
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
```

```
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
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']
```

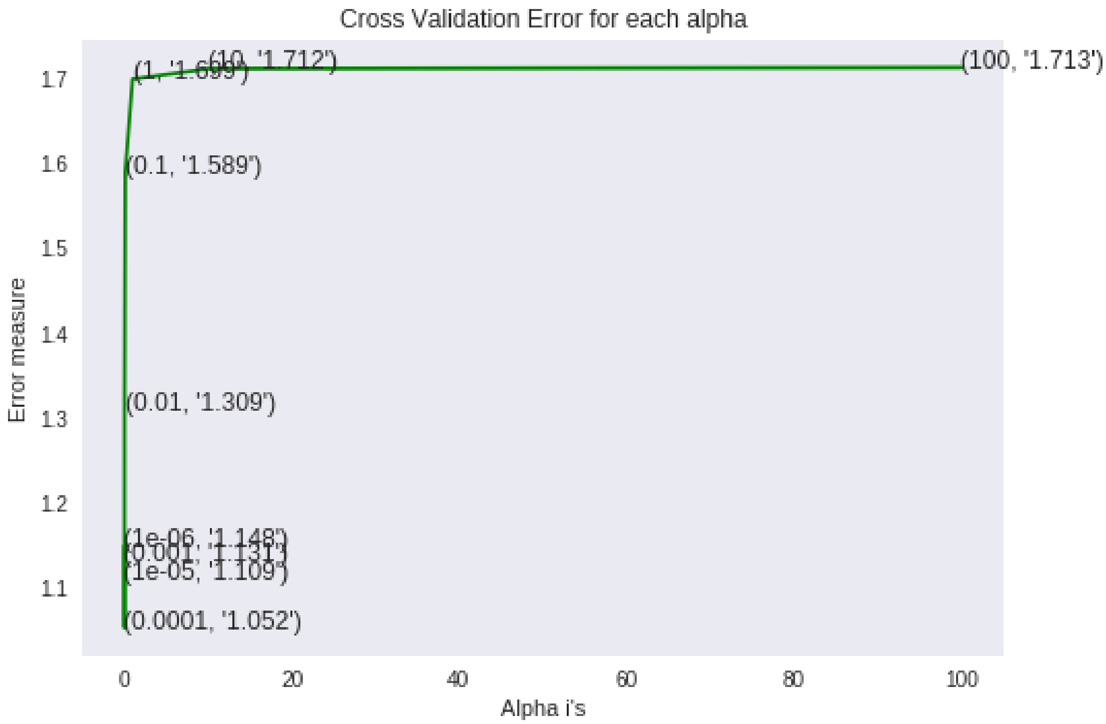
```
Predicted Class : 7
Predicted Class Probabilities: [[6.200e-03 1.185e-01 9.000e-03 9.900e-03 1.050e-02 4.690e-02 7
  3.400e-03 7.000e-04]]
Actual Class : 7
--------------------------------------------------
10 Text feature [cells] present in test data point [True]
462 Text feature [cyclin] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

5.0.3 Without class balancing

```
In [0]: alpha = [10 ** x for x in range(-6, 3)]
        cv_log_error_array = []
        for i in alpha:
            print("for alpha =", i)
            #clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear'
            clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
            clf.fit(train_x_tfidfCoding, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x_tfidfCoding, train_y)
            sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
            cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1
            # to avoid rounding error while multiplying probabilites we use log-probability es
            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 = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear')
        clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
        clf.fit(train_x_tfidfCoding, train_y)
```

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.1482985716161132
for alpha = 1e-05
Log Loss : 1.1087943166869167
for alpha = 0.0001
Log Loss : 1.052124126266507
for alpha = 0.001
Log Loss : 1.1305748208232422
for alpha = 0.01
Log Loss : 1.3086997576477362
for alpha = 0.1
Log Loss : 1.5894484908433661
for alpha = 1
Log Loss : 1.6994349945017733
for alpha = 10
Log Loss : 1.7117163554187154
for alpha = 100
Log Loss : 1.7130709644181807
```



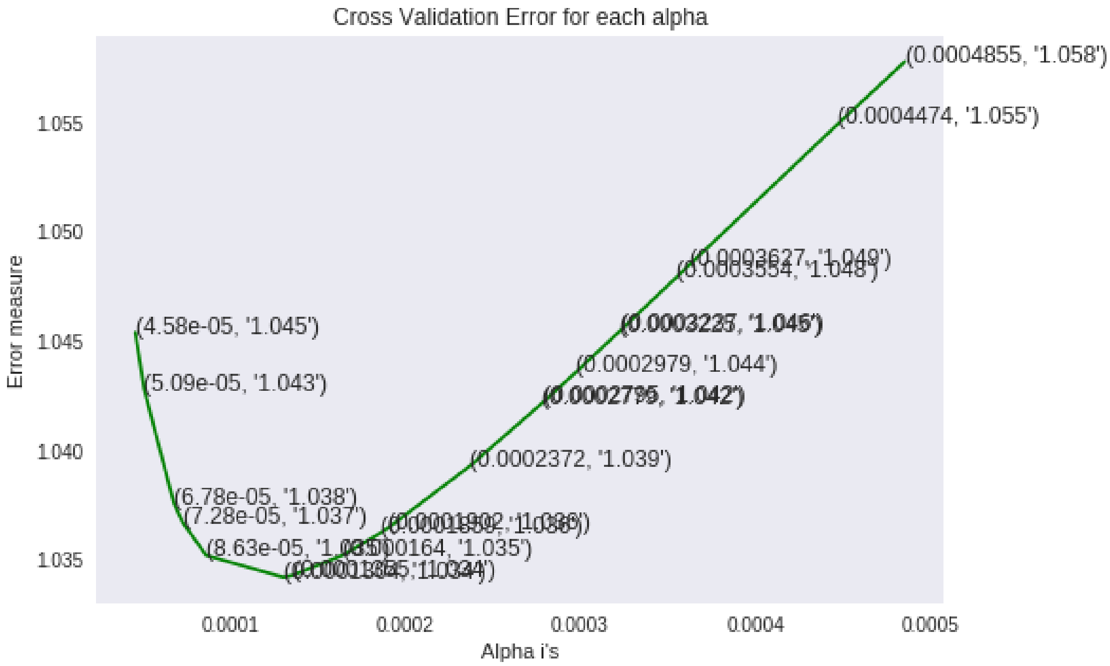

```
For values of best alpha =  0.0001 The train log loss is: 0.4315008587419229
For values of best alpha =  0.0001 The cross validation log loss is: 1.052124126266507
For values of best alpha =  0.0001 The test log loss is: 1.0853477904936566
```

In [26]:
```
alpha = np.random.uniform(0.00002,0.0005,20)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    # to avoid rounding error while multiplying probabilites we use log-probability e
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
```

```
        ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
    plt.grid()
    plt.title("Cross Validation Error for each alpha")
    plt.xlabel("Alpha i's")
    plt.ylabel("Error measure")
    plt.show()


    best_alpha = np.argmin(cv_log_error_array)
    #clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear')
    clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
for alpha = 4.58e-05
Log Loss : 1.045395395421416
for alpha = 5.09e-05
Log Loss : 1.0427808336164082
for alpha = 6.78e-05
Log Loss : 1.0375930257185433
for alpha = 7.28e-05
Log Loss : 1.036721144352279
for alpha = 8.63e-05
Log Loss : 1.0352052592227805
for alpha = 0.0001304
Log Loss : 1.0342059271610662
for alpha = 0.0001355
Log Loss : 1.0342942856146393
for alpha = 0.000164
Log Loss : 1.0352010753785488
for alpha = 0.0001859
Log Loss : 1.0362302190052115
for alpha = 0.0001902
Log Loss : 1.0364566643610214
for alpha = 0.0002372
Log Loss : 1.0393033886710636
for alpha = 0.0002779
Log Loss : 1.0421382305488291
for alpha = 0.0002795
Log Loss : 1.0422541362031614
```

```
for alpha = 0.0002979
Log Loss : 1.0436041723756635
for alpha = 0.0003225
Log Loss : 1.0454462434068137
for alpha = 0.0003237
Log Loss : 1.0455368770071063
for alpha = 0.0003554
Log Loss : 1.0479463515719536
for alpha = 0.0003627
Log Loss : 1.048503668637081
for alpha = 0.0004474
Log Loss : 1.0549347462444285
for alpha = 0.0004855
Log Loss : 1.0577674510451391
```



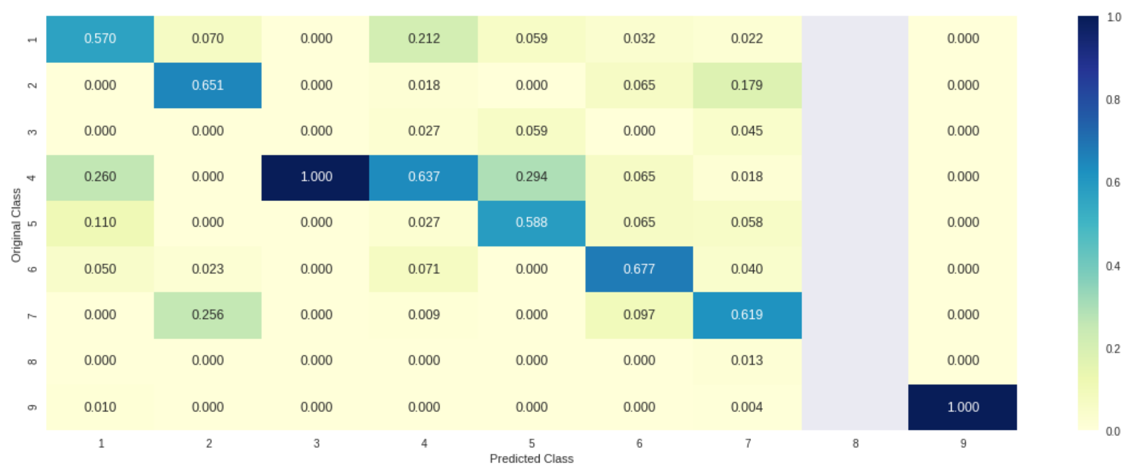

```
For values of best alpha =  0.0001304 The train log loss is: 0.4589573283729896
For values of best alpha =  0.0001304 The cross validation log loss is: 1.0342059271610662
For values of best alpha =  0.0001304 The test log loss is: 1.0634787224064821
```

```
In [27]: #testing
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

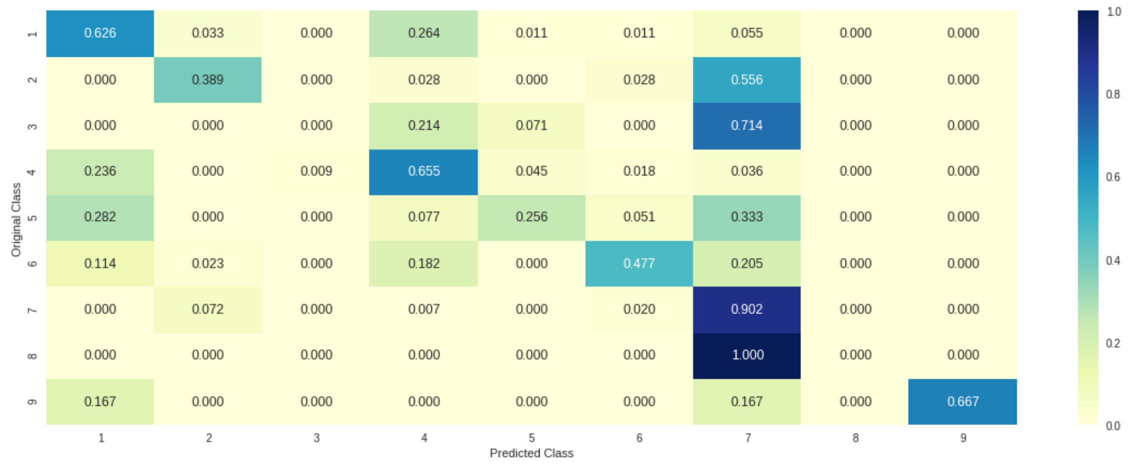
```
Log loss : 1.0342059271610662
Number of mis-classified points : 0.37969924812030076
```

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

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



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

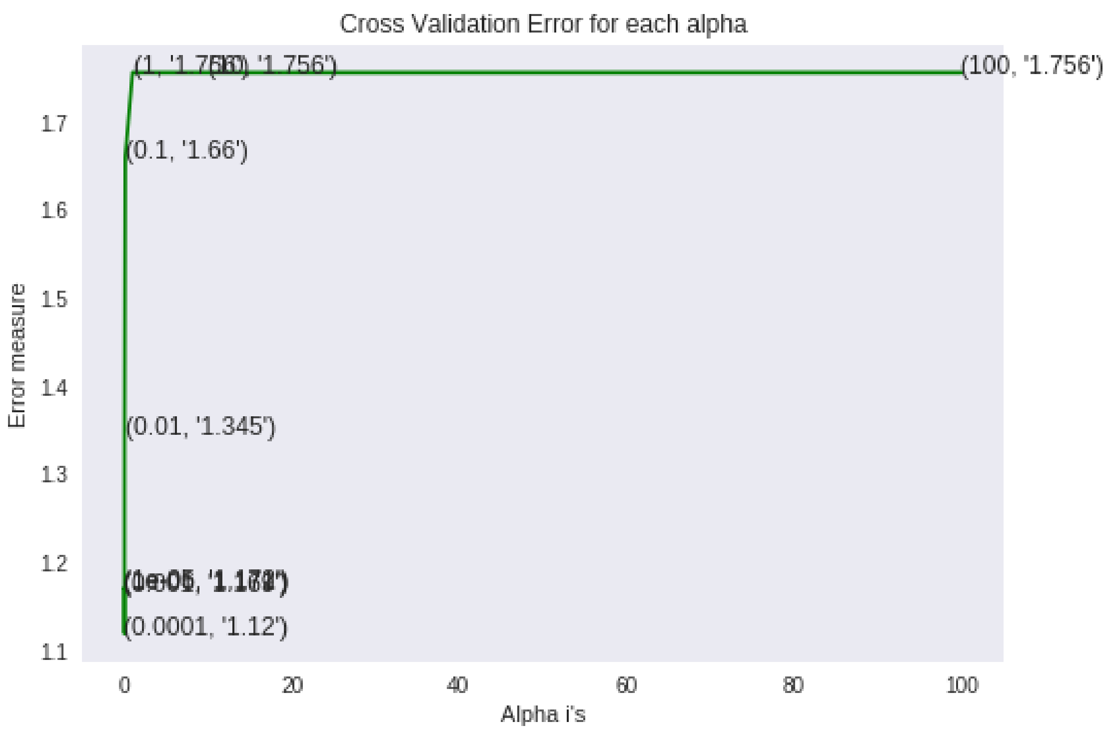


6 Linear SVMs

In [0]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42,class_wei
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    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 es
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```python
        sig_clf.fit(train_x_tfidfCoding, train_y)

        predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
        predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.1706309531961865
for alpha = 1e-05
Log Loss : 1.1718508993311973
for alpha = 0.0001
Log Loss : 1.119577365758023
for alpha = 0.001
Log Loss : 1.1687927693866187
for alpha = 0.01
Log Loss : 1.3450141021099664
for alpha = 0.1
Log Loss : 1.6597066814482349
for alpha = 1
Log Loss : 1.755571299775473
for alpha = 10
Log Loss : 1.755571300253828
for alpha = 100
Log Loss : 1.7555713023916553
```

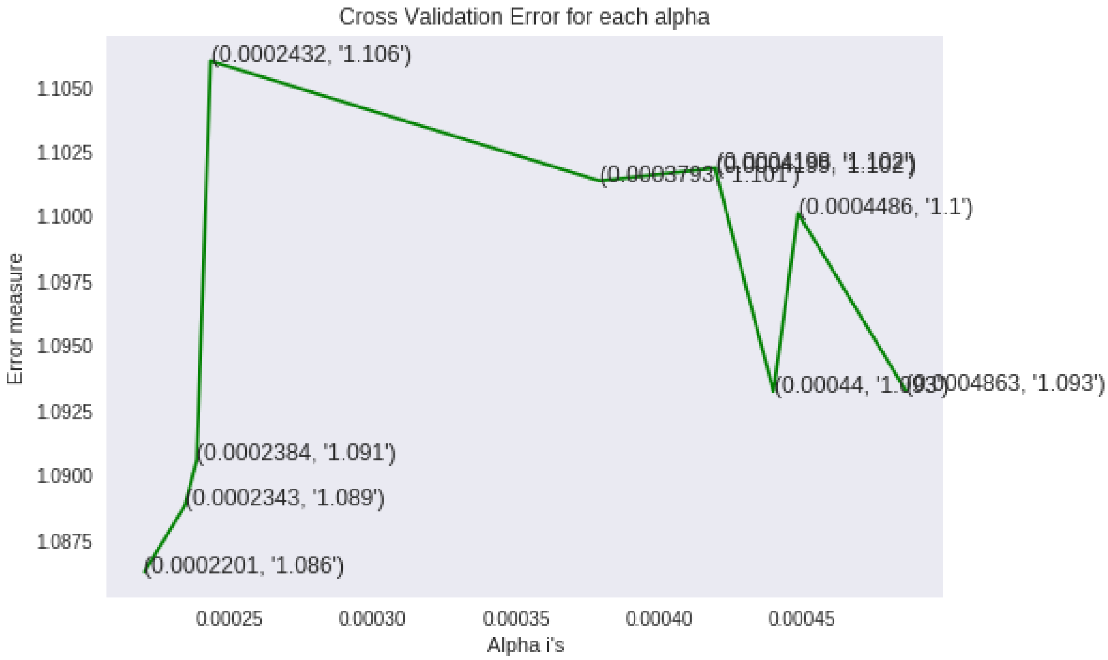



```
For values of best alpha =  0.0001 The train log loss is: 0.4728529643370944
For values of best alpha =  0.0001 The cross validation log loss is: 1.1401692726762
For values of best alpha =  0.0001 The test log loss is: 1.1638518010544059
```

In [28]:
```
alpha = np.random.uniform(0.0002,0.0005,10)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42,class_wei
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    # to avoid rounding error while multiplying probabilites we use log-probability e
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
```

```
    plt.grid()
    plt.title("Cross Validation Error for each alpha")
    plt.xlabel("Alpha i's")
    plt.ylabel("Error measure")
    plt.show()


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

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

```
for alpha = 0.0002201
Log Loss : 1.0862762207699164
for alpha = 0.0002343
Log Loss : 1.088838013034623
for alpha = 0.0002384
Log Loss : 1.0905946310129409
for alpha = 0.0002432
Log Loss : 1.1060022312558488
for alpha = 0.0003793
Log Loss : 1.1013698764108202
for alpha = 0.0004198
Log Loss : 1.1018688036597901
for alpha = 0.0004199
Log Loss : 1.101712274163335
for alpha = 0.00044
Log Loss : 1.0932409468310254
for alpha = 0.0004486
Log Loss : 1.1001268274109268
for alpha = 0.0004863
Log Loss : 1.0932668361513767
```

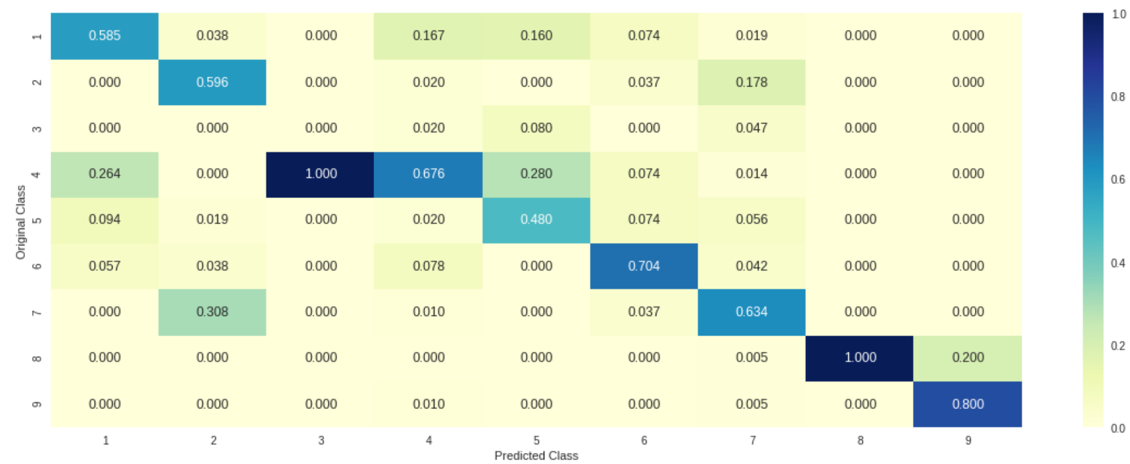



```
For values of best alpha =  0.0002201 The train log loss is: 0.48235993835298313
For values of best alpha =  0.0002201 The cross validation log loss is: 1.0862762207699164
For values of best alpha =  0.0002201 The test log loss is: 1.1270212318445756
```

```
In [29]: ##testing
         clf = SGDClassifier(alpha=0.0002201, penalty='l2', loss='hinge', random_state=42,class
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

```
Log loss : 1.0862762207699164
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```

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



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

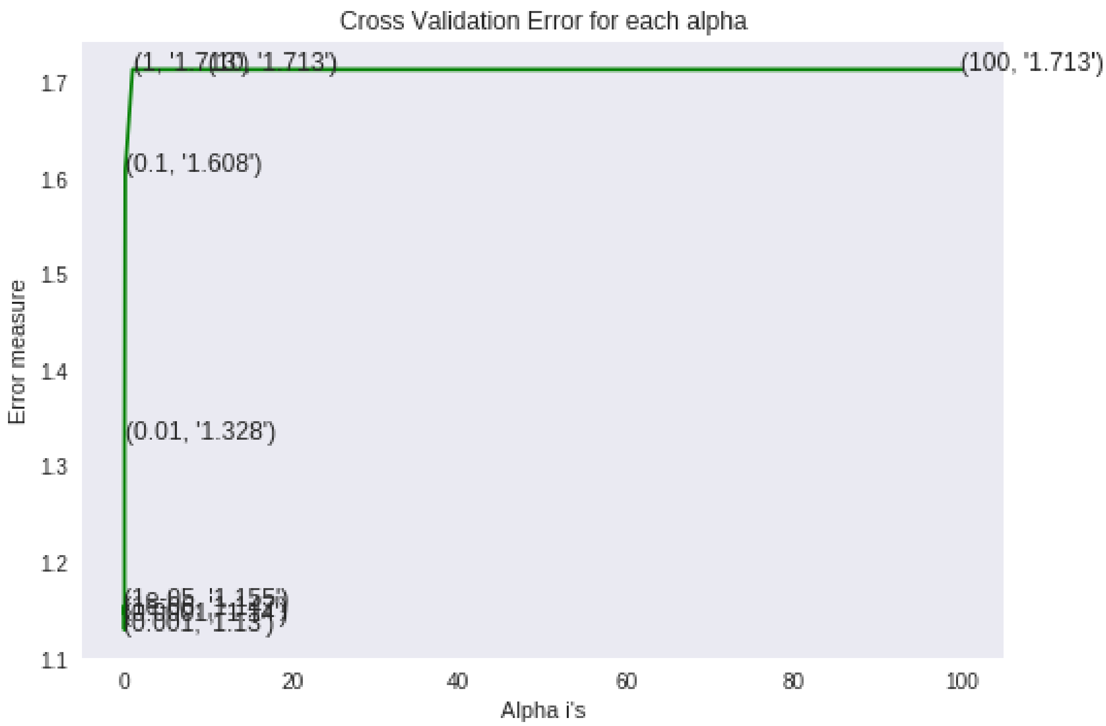
6.0.1 Without class balancing

```
In [0]: alpha = [10 ** x for x in range(-6, 3)]
        #alpha = np.random.uniform(0.00005,0.0005,15)
        #alpha = np.round(alpha,7)
        #alpha.sort()
```

```
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    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 es
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_lo
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.1470974714657551
for alpha = 1e-05
Log Loss : 1.1551738406235221
for alpha = 0.0001
Log Loss : 1.1401692726762
for alpha = 0.001
Log Loss : 1.1302116887789937
for alpha = 0.01
Log Loss : 1.3283703655044943
for alpha = 0.1
Log Loss : 1.6083674634040097
```

```
for alpha = 1
Log Loss : 1.7133831610096346
for alpha = 10
Log Loss : 1.713383093398969
for alpha = 100
Log Loss : 1.7133831773758716
```



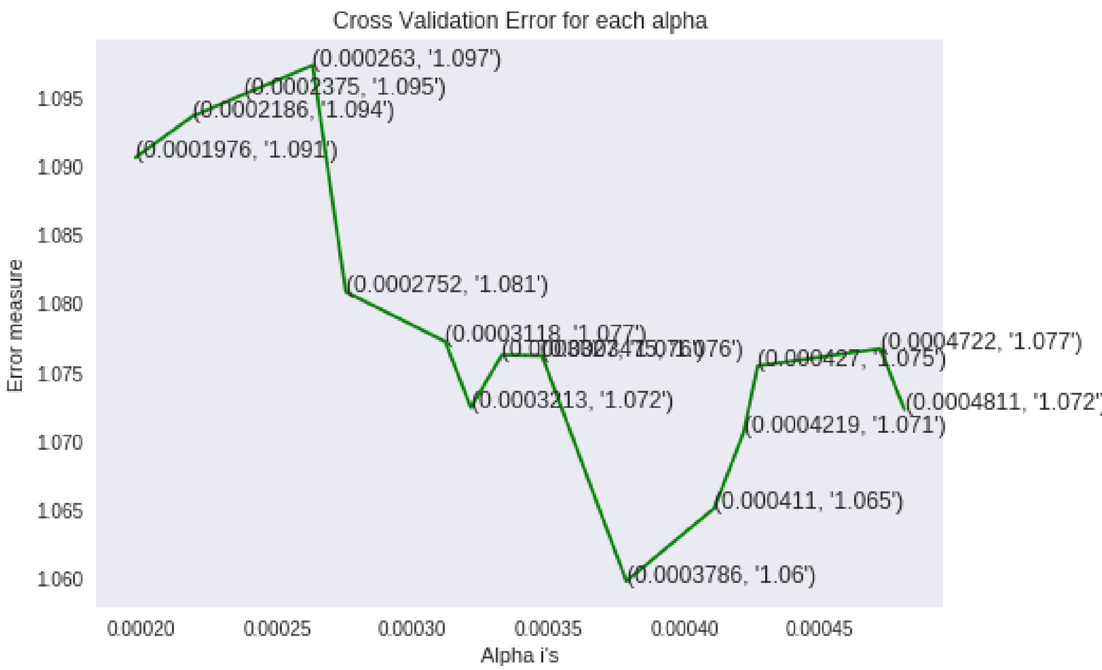

```
For values of best alpha =  0.001 The train log loss is: 0.5527913676115366
For values of best alpha =  0.001 The cross validation log loss is: 1.1302116887789937
For values of best alpha =  0.001 The test log loss is: 1.1671880366993928
```

In [30]:
```
alpha = np.random.uniform(0.0001,0.0005,15)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
```

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

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


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

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

```
for alpha = 0.0001976
Log Loss : 1.0906588768480132
for alpha = 0.0002186
Log Loss : 1.093656241490354
for alpha = 0.0002375
Log Loss : 1.0952650711331624
for alpha = 0.000263
Log Loss : 1.097391321480177
for alpha = 0.0002752
Log Loss : 1.0808679767297498
for alpha = 0.0003118
Log Loss : 1.0772581459533717
for alpha = 0.0003213
Log Loss : 1.0724358018603268
for alpha = 0.0003327
Log Loss : 1.0762586514712817
for alpha = 0.0003475
Log Loss : 1.0762203681368245
for alpha = 0.0003786
Log Loss : 1.0597794689167068
```

```
for alpha = 0.000411
Log Loss : 1.0651000724105288
for alpha = 0.0004219
Log Loss : 1.0706589665699928
for alpha = 0.000427
Log Loss : 1.0754733244998016
for alpha = 0.0004722
Log Loss : 1.0767283943428376
for alpha = 0.0004811
Log Loss : 1.0722856380539214
```



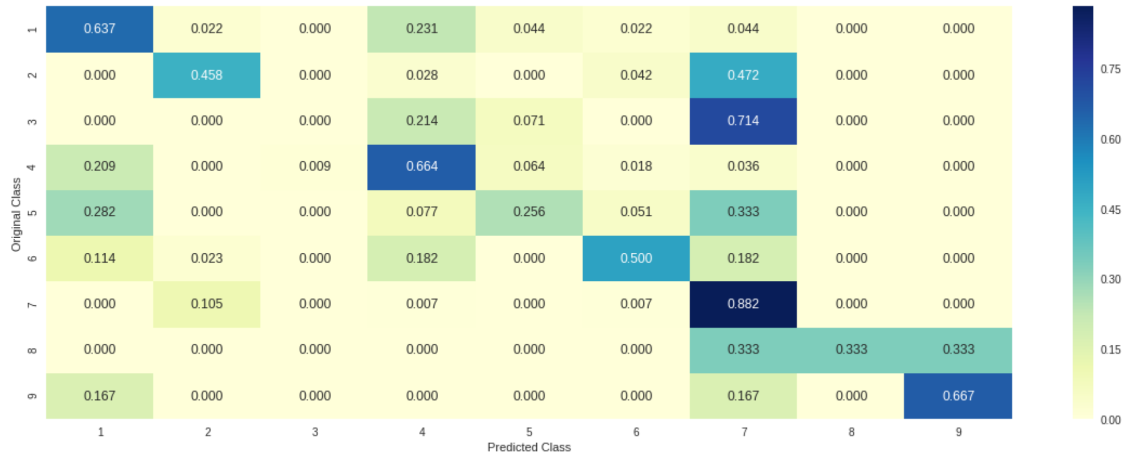

```
For values of best alpha =  0.0003786 The train log loss is: 0.44668092325855907
For values of best alpha =  0.0003786 The cross validation log loss is: 1.0597794689167068
For values of best alpha =  0.0003786 The test log loss is: 1.1044631464817687
```

In [31]:
```
##testing
clf = SGDClassifier(alpha=0.0003786, penalty='l2', loss='hinge', random_state=42)
predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

```
Log loss : 1.0597794689167068
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```

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

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



6.0.2 Feature importance

```
In [0]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
        clf.fit(train_x_tfidfCoding,train_y)
        test_point_index = 1
        # test_point_index = 100
        no_feature = 500
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
        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 : 7
Predicted Class Probabilities: [[0.0402 0.0152 0.0095 0.0295 0.1212 0.0278 0.632  0.0053 0.1193
Actual Class : 7
--------------------------------------------------
0 Text feature [cells] present in test data point [True]
16 Text feature [of] present in test data point [True]
229 Text feature [kit] present in test data point [True]
396 Text feature [crizotinib] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

```
In [0]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
        clf.fit(train_x_tfidfCoding,train_y)
        test_point_index = 94
        no_feature = 500
        predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0649 0.0415 0.0271 0.7866 0.015  0.0212 0.0389 0.0015 0.0032
Actual Class : 4
--------------------------------------------------
199 Text feature [protein] present in test data point [True]
462 Text feature [and] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

7 Random Forest Classifier

In [0]:
```
alpha = [100,200,500,1000,2000]
max_depth = [5,10,20]
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, ran
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, ep
        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_er
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/3)], criterion='gini', m
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
        sig_clf.fit(train_x_tfidfCoding, train_y)

        predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/3)],'depth = ',alpha[int
        predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/3)],'depth = ',alpha[int
        predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
        print('For values of best estimator = ', alpha[int(best_alpha/3)],'depth = ',alpha[int
```

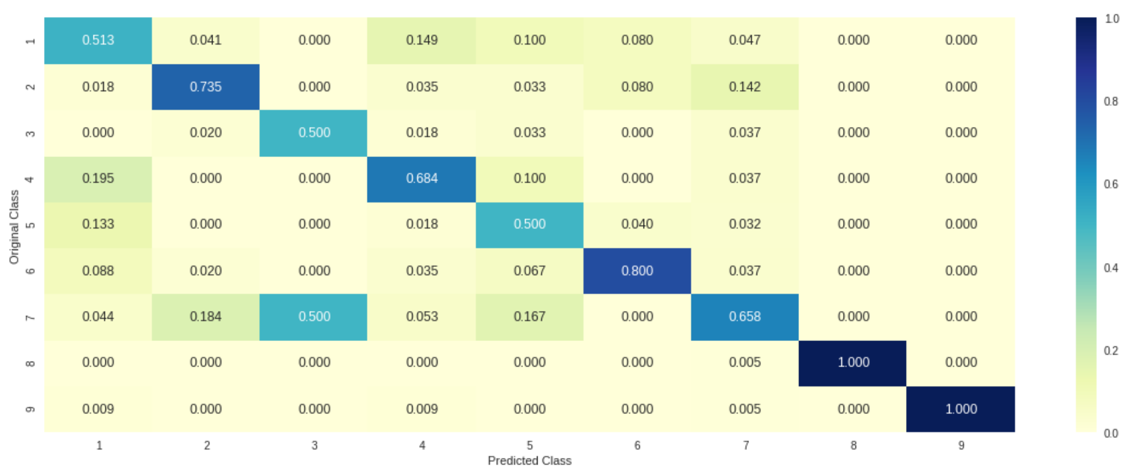
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2287377443960394
for n_estimators = 100 and max depth =  10
Log Loss : 1.1292795926475636
for n_estimators = 100 and max depth =  20
Log Loss : 1.0735000875010572
for n_estimators = 200 and max depth =  5
Log Loss : 1.2162306766553685
for n_estimators = 200 and max depth =  10
Log Loss : 1.1220066223884533
for n_estimators = 200 and max depth =  20
Log Loss : 1.0765072271284166
for n_estimators = 500 and max depth =  5
Log Loss : 1.2035459020894086
for n_estimators = 500 and max depth =  10
Log Loss : 1.112493546288718
for n_estimators = 500 and max depth =  20
Log Loss : 1.0689112413766457
for n_estimators = 1000 and max depth =  5
Log Loss : 1.204816478598033
for n_estimators = 1000 and max depth =  10
Log Loss : 1.113580707652912
for n_estimators = 1000 and max depth =  20
Log Loss : 1.071608316765358
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2021403696536532
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1132095015893888
for n_estimators = 2000 and max depth =  20
Log Loss : 1.071239133154997
For values of best estimator =  500 depth =  500 The train log loss is: 0.5032411219015764
For values of best estimator =  500 depth =  500 The cross validation log loss is: 1.06891124l3
For values of best estimator =  500 depth =  500 The test log loss is: 1.115363178802882
```

```
In [32]: #test
         clf = RandomForestClassifier(n_estimators=500, criterion='gini', max_depth=20, random_
         predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding,cv_y,
```

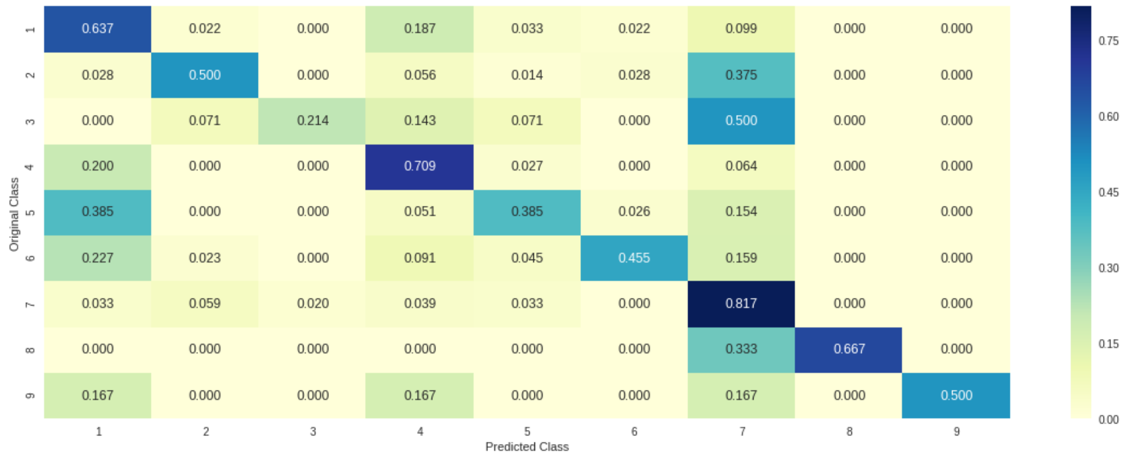
```
Log loss : 1.0296946713555744
```

```
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```


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




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



7.0.1 Feature Importance

```
In [0]: clf = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=20, random_s
        clf.fit(train_x_tfidfCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidfCoding, train_y)

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

```
Predicted Class : 9
Predicted Class Probabilities: [[0.0234 0.028  0.0103 0.0274 0.0257 0.0198 0.3803 0.0034 0.4817
Actual Class : 7
--------------------------------------------------
0 Text feature [cell] present in test data point [True]
1 Text feature [protein] present in test data point [True]
3 Text feature [cells] present in test data point [True]
4 Text feature [resistance] present in test data point [True]
6 Text feature [of] present in test data point [True]
8 Text feature [to] present in test data point [True]
9 Text feature [in] present in test data point [True]
11 Text feature [and] present in test data point [True]
15 Text feature [the] present in test data point [True]
20 Text feature [kit] present in test data point [True]
21 Text feature [et] present in test data point [True]
```

```
34 Text feature [alleles] present in test data point [True]
58 Text feature [melanoma] present in test data point [True]
83 Text feature [gastric] present in test data point [True]
97 Text feature [endometrial] present in test data point [True]
Out of the top  100  features  15 are present in query point
```

In [0]:
```
test_point_index = 52
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCodi
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0223 0.0268 0.0122 0.0218 0.0293 0.8564 0.0231 0.0038 0.0044
Actual Class : 6
--------------------------------------------------
0 Text feature [cell] present in test data point [True]
1 Text feature [protein] present in test data point [True]
2 Text feature [variants] present in test data point [True]
3 Text feature [cells] present in test data point [True]
4 Text feature [resistance] present in test data point [True]
5 Text feature [patients] present in test data point [True]
6 Text feature [of] present in test data point [True]
7 Text feature [deleterious] present in test data point [True]
8 Text feature [to] present in test data point [True]
9 Text feature [in] present in test data point [True]
11 Text feature [and] present in test data point [True]
13 Text feature [ovarian] present in test data point [True]
15 Text feature [the] present in test data point [True]
17 Text feature [p53] present in test data point [True]
19 Text feature [families] present in test data point [True]
21 Text feature [et] present in test data point [True]
34 Text feature [alleles] present in test data point [True]
Out of the top  100  features  17 are present in query point
```

8 Stacking the models

In [0]:
```
clf1 = SGDClassifier(alpha=0.0001328, penalty='l2', loss='log', class_weight='balanced
clf1.fit(train_x_tfidfCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.000371, penalty='l2', loss='hinge', random_state=0)
```

```
        clf2.fit(train_x_tfidfCoding, train_y)
        sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


        clf3 = KNeighborsClassifier(n_neighbors=5)
        clf3.fit(train_x_tfidfCoding, train_y)
        sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

        sig_clf1.fit(train_x_tfidfCoding, train_y)
        print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba
        sig_clf2.fit(train_x_tfidfCoding, train_y)
        print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_p
        sig_clf3.fit(train_x_tfidfCoding, train_y)
        print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tf
        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_classif
            sclf.fit(train_x_tfidfCoding, train_y)
            print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_l
            log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
            if best_alpha > log_error:
                best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.09
Naive Bayes : Log Loss: 1.18
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.175
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.009
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.452
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.109
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.229
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.605
```

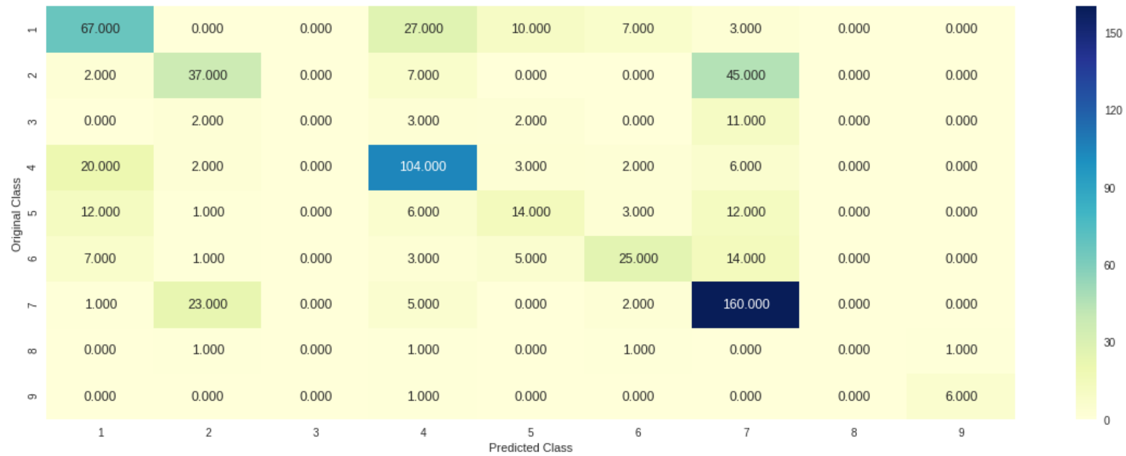
```
In [0]: #alpha = [0.0001,0.001,0.01,0.1,1,10]
        alpha = np.random.uniform(0.005,0.5,10)
        alpha = np.round(alpha,5)
        alpha.sort()
        best_alpha = 999
        for i in alpha:
            lr = LogisticRegression(C=i)
            sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classif
            sclf.fit(train_x_tfidfCoding, train_y)
            print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_l
```

```
            log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
            if best_alpha > log_error:
                best_alpha = log_error
```

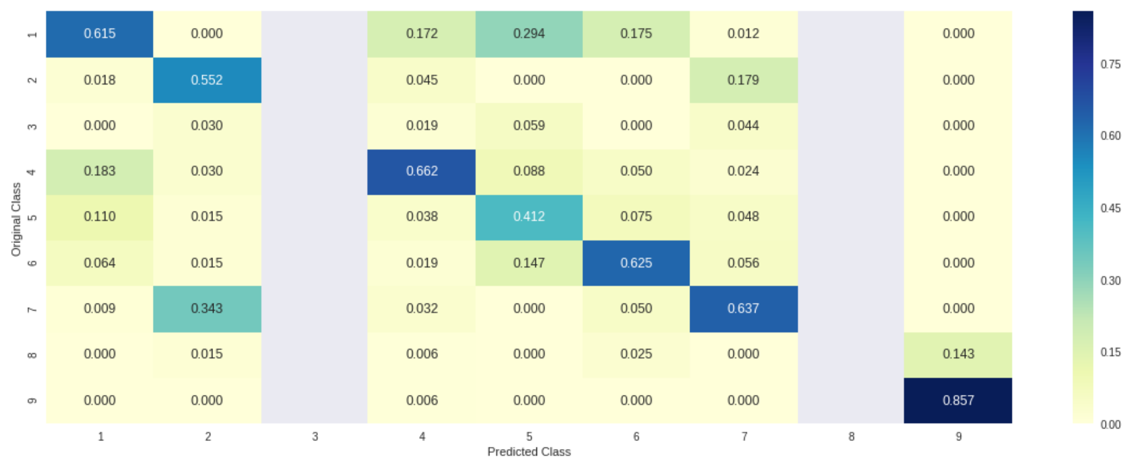
```
Stacking Classifer : for the value of alpha: 0.102980 Log Loss: 1.108
Stacking Classifer : for the value of alpha: 0.115560 Log Loss: 1.105
Stacking Classifer : for the value of alpha: 0.120760 Log Loss: 1.104
Stacking Classifer : for the value of alpha: 0.128970 Log Loss: 1.103
Stacking Classifer : for the value of alpha: 0.224900 Log Loss: 1.110
Stacking Classifer : for the value of alpha: 0.264670 Log Loss: 1.116
Stacking Classifer : for the value of alpha: 0.333400 Log Loss: 1.128
Stacking Classifer : for the value of alpha: 0.337090 Log Loss: 1.129
Stacking Classifer : for the value of alpha: 0.337210 Log Loss: 1.129
Stacking Classifer : for the value of alpha: 0.433760 Log Loss: 1.146
```

In [0]:
```
#testing
lr = LogisticRegression(C=0.128970)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=l
sclf.fit(train_x_tfidfCoding, train_y)

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

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

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

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

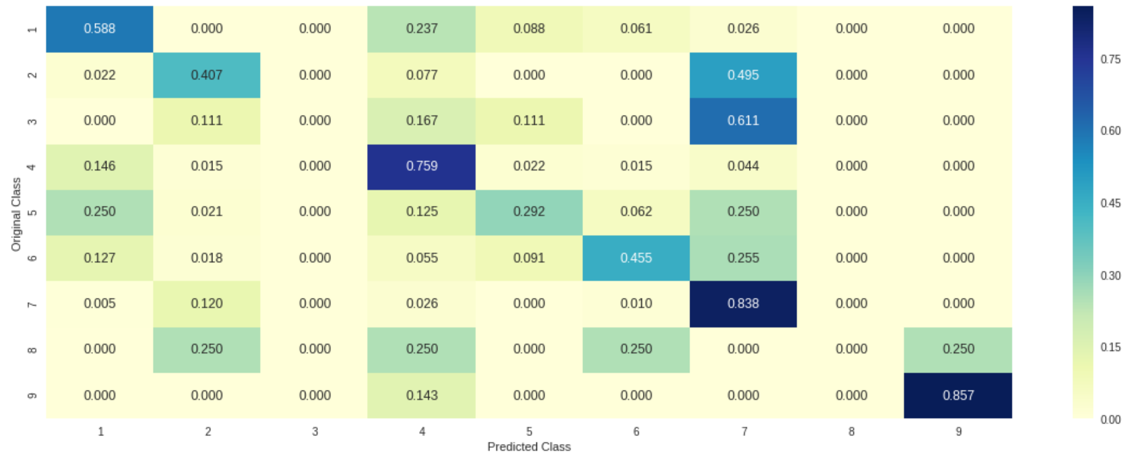
```
Log loss (train) on the stacking classifier : 0.3640556307316504
Log loss (CV) on the stacking classifier : 1.103382586268113
Log loss (test) on the stacking classifier : 1.1823532938143695
Number of missclassified point : 0.37894736842105264
-------------------- Confusion matrix --------------------
```



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



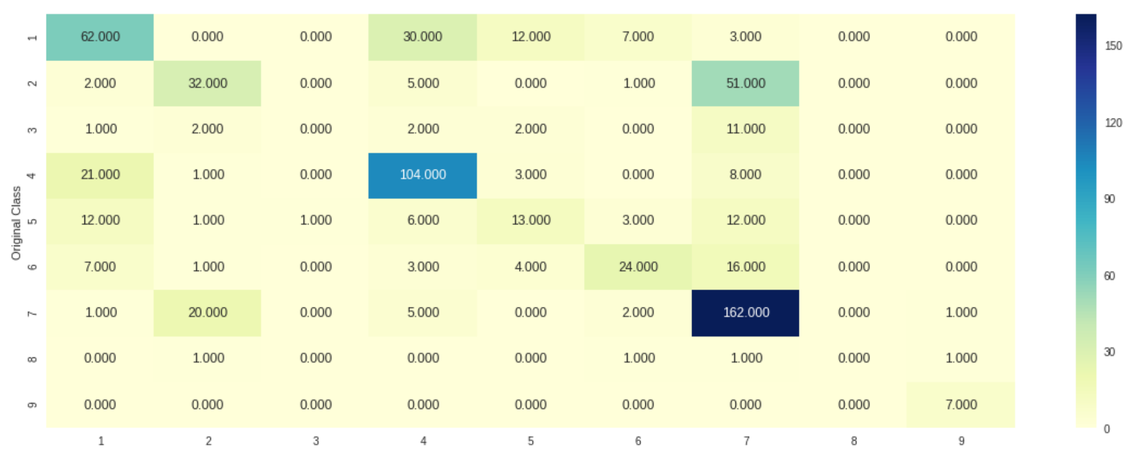
-------------------- Recall matrix (Row sum=1) --------------------



8.1 Maximum Voting Classifier

```
In [0]: from sklearn.ensemble import VotingClassifier
        vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_cl
        vclf.fit(train_x_tfidfCoding, train_y)
        print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_prol
        print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_
        print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba
        print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfidfCo
        plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidfCoding))
```

```
Log loss (train) on the VotingClassifier : 0.5542565795910458
Log loss (CV) on the VotingClassifier : 1.0703292269713045
Log loss (test) on the VotingClassifier : 1.11206377381788
Number of missclassified point : 0.3924812030075188
-------------------- Confusion matrix --------------------
```



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


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

9 Conclusion

	Train loss	CV loss	Test loss
Naive-Bayes	0.763	1.204	1.199

	Train loss	CV loss	Test loss
K-NN	0.986	1.160	1.216
Logistic Regression(With Class Balancing)	0.444	1.058	1.080
Logistic Regression(Without Class Balancing)	0.431	1.052	1.085
Linear SVM(With Class Balancing)	0.462	1.104	1.159
Linear SVM(Without Class Balancing)	0.450	1.099	1.216
Random Forest	0.503	1.068	1.115