CancerDiagnosis-Assignment-3

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]: # 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]: #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]: 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
```

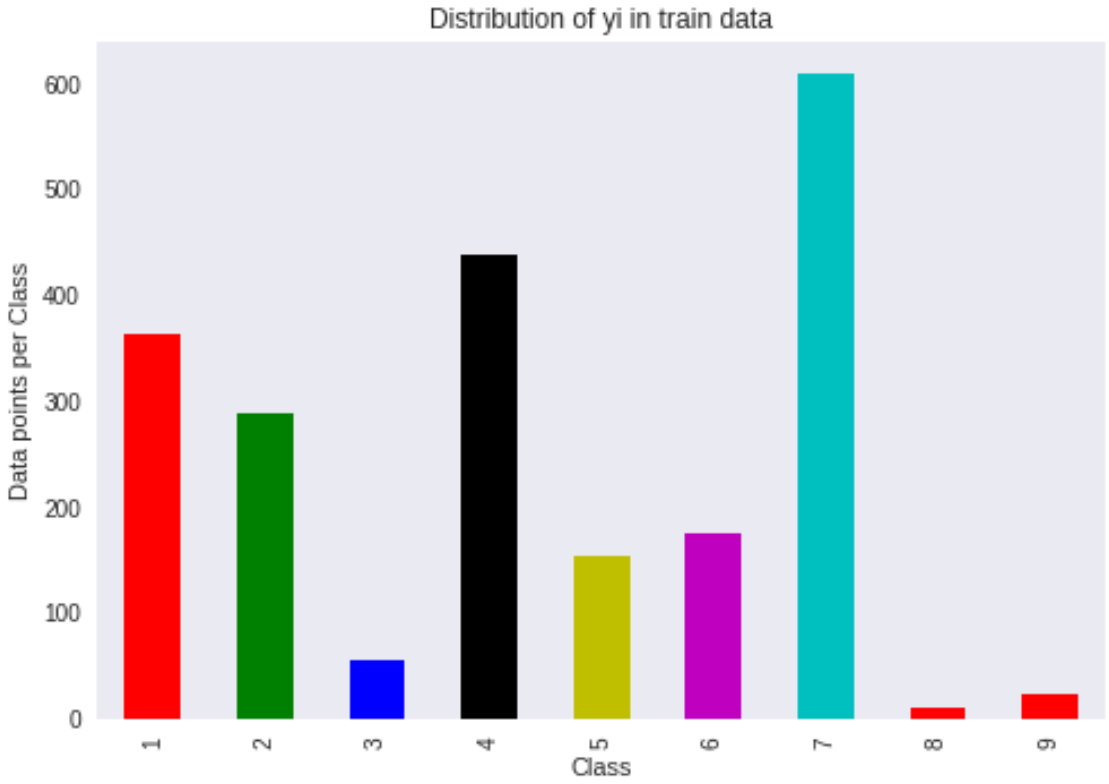
```
In [12]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

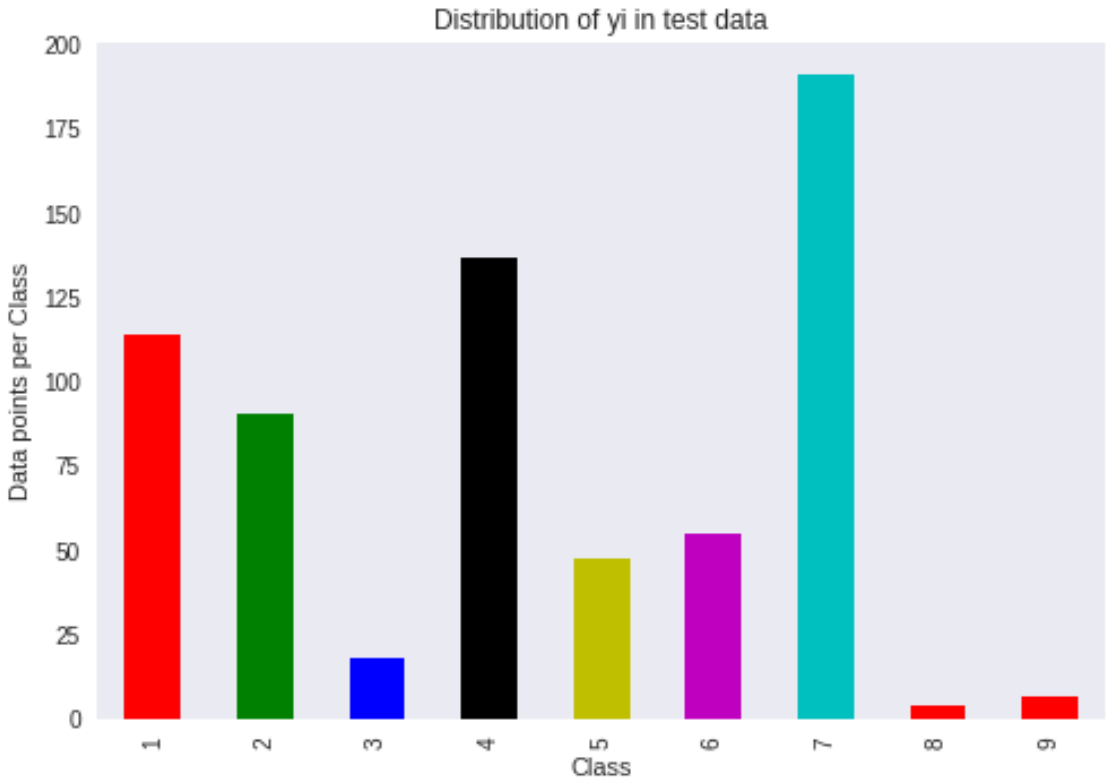
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
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[


         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']
```

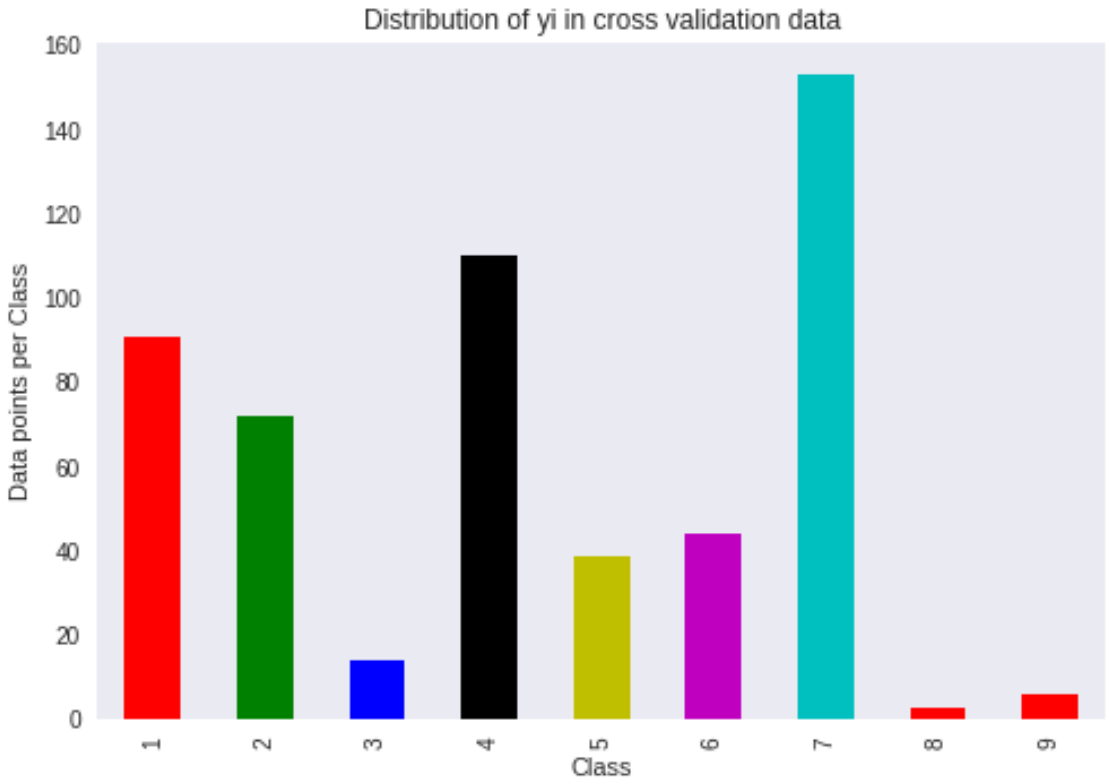
```
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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 Bag of words(Unigrams-Bigrams)

```
In [0]: gene_vectorizer = CountVectorizer()
        train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
        test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
        cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

```
In [0]: variation_vectorizer = CountVectorizer()
        train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
        test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
        cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [16]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'].value
# getting all the feature names (words)
train_text_features   = text_vectorizer.get_feature_names()

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

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


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

# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```
Total number of unique words in train data : 684163
```

2.0.1 Stacking Features

In [0]:
```
#Bow
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_o

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

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

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

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, yticklabel
            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()
```

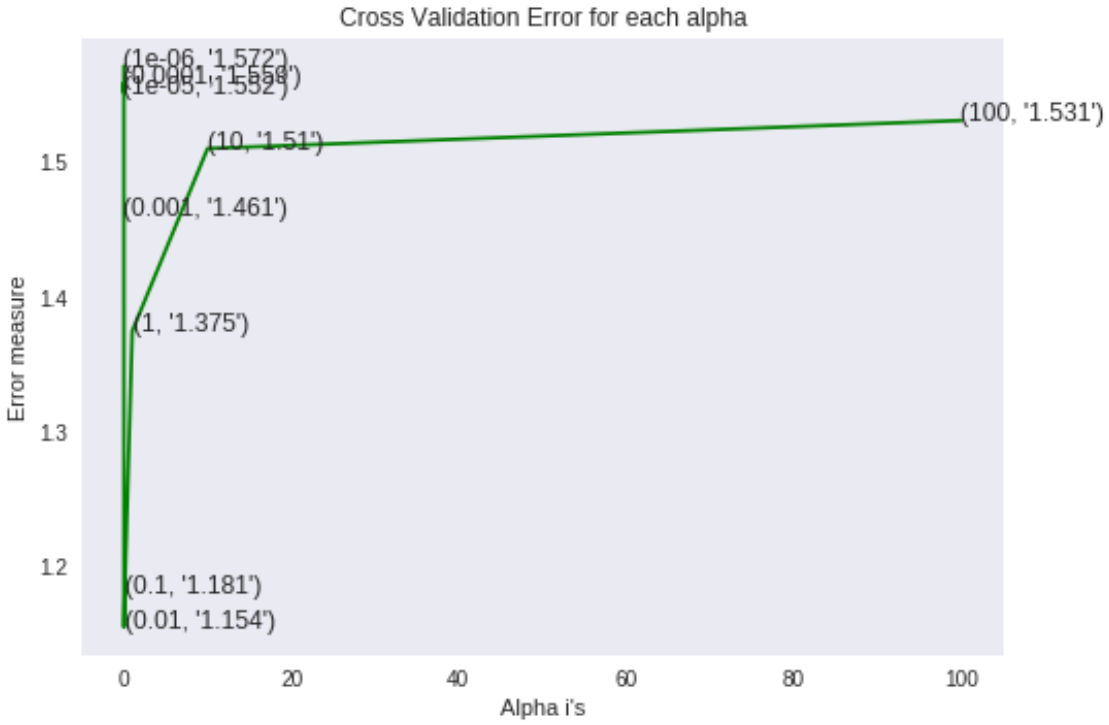
```
In [0]: #Data preparation for ML models.

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

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

3.1 Logistic Regression

```
In [20]: alpha = [10 ** x for x in range(-6, 3)]
         cv_log_error_array = []
         for i in alpha:
             print("for alpha =", i)
             clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
             clf.fit(train_x_onehotCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_onehotCoding, train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
             # to avoid rounding error while multiplying probabilites we use log-probability e
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for alpha = 1e-06
Log Loss : 1.571614707421731
for alpha = 1e-05
Log Loss : 1.5519803140048272
for alpha = 0.0001
Log Loss : 1.5591862865226462
for alpha = 0.001
Log Loss : 1.461079498854767
for alpha = 0.01
Log Loss : 1.154397979609746
for alpha = 0.1
Log Loss : 1.1805530631591723
for alpha = 1
Log Loss : 1.3745867487547532
for alpha = 10
Log Loss : 1.5103044947090372
for alpha = 100
Log Loss : 1.5311654908430445
```



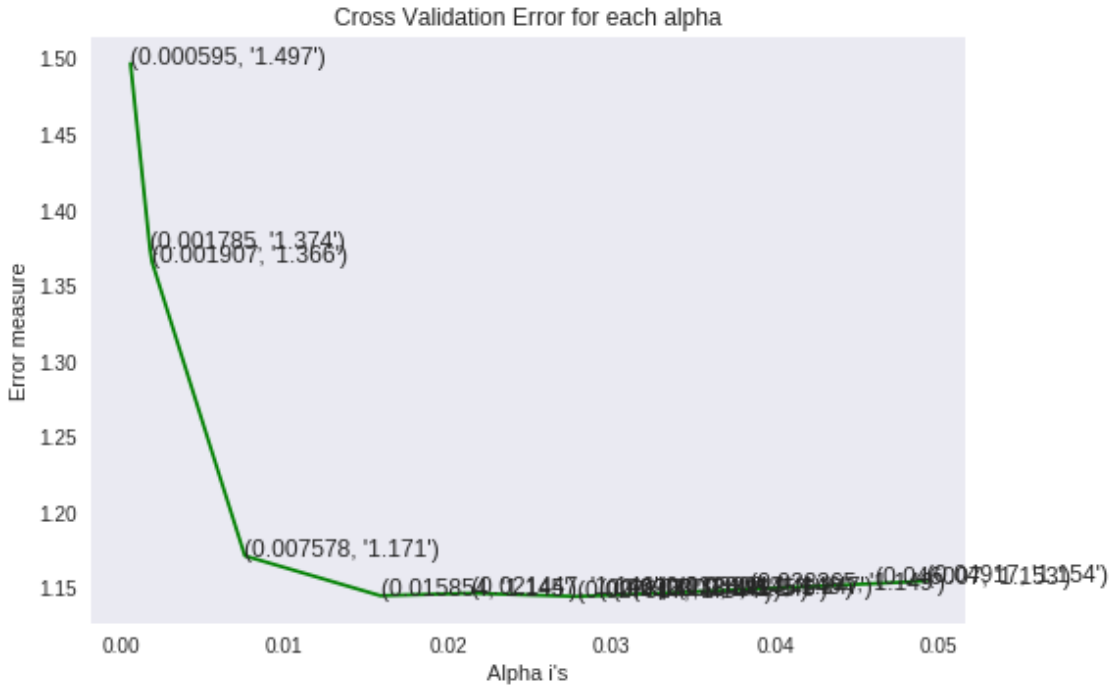

```
For values of best alpha =  0.01 The train log loss is: 0.8637076433122415
For values of best alpha =  0.01 The cross validation log loss is: 1.154397979609746
For values of best alpha =  0.01 The test log loss is: 1.1493210437721706
```

```
In [21]: alpha = np.random.uniform(0.0005,0.05,15)
         alpha = np.round(alpha,6)
         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_onehotCoding  , train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
             sig_clf.fit(train_x_onehotCoding  , train_y)
             sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding  )
             cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
             # to avoid rounding error while multiplying probabilites we use log-probability e
             print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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


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

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

```
for alpha = 0.000595
Log Loss : 1.4969580136052472
for alpha = 0.001785
Log Loss : 1.374264436149108
for alpha = 0.001907
Log Loss : 1.3657171597947546
for alpha = 0.007578
Log Loss : 1.1711334008485925
for alpha = 0.015854
Log Loss : 1.1447751215964141
for alpha = 0.021447
Log Loss : 1.1464351073814603
for alpha = 0.027817
Log Loss : 1.1442794503372584
for alpha = 0.02923
Log Loss : 1.1448220677582406
for alpha = 0.030007
Log Loss : 1.145134636001083
for alpha = 0.031184
Log Loss : 1.1456605860163758
for alpha = 0.032809
Log Loss : 1.1463982204717609
for alpha = 0.033917
Log Loss : 1.146904219539773
for alpha = 0.038365
Log Loss : 1.149046607944418
for alpha = 0.046007
Log Loss : 1.1527990483193513
```

```
for alpha = 0.04917
Log Loss : 1.1544030445514355
```

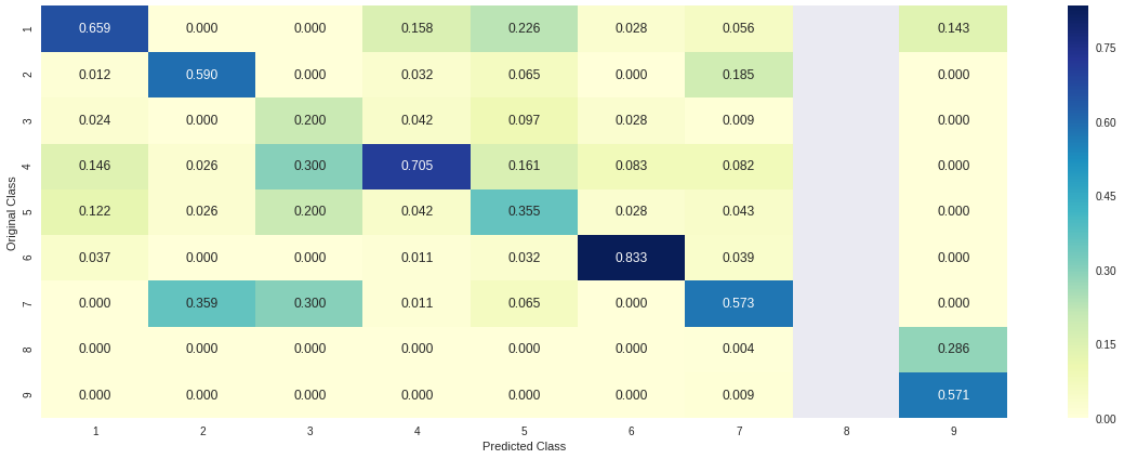



```
For values of best alpha =  0.027817 The train log loss is: 0.827283284056054
For values of best alpha =  0.027817 The cross validation log loss is: 1.1442794503372584
For values of best alpha =  0.027817 The test log loss is: 1.1304407389334683
```

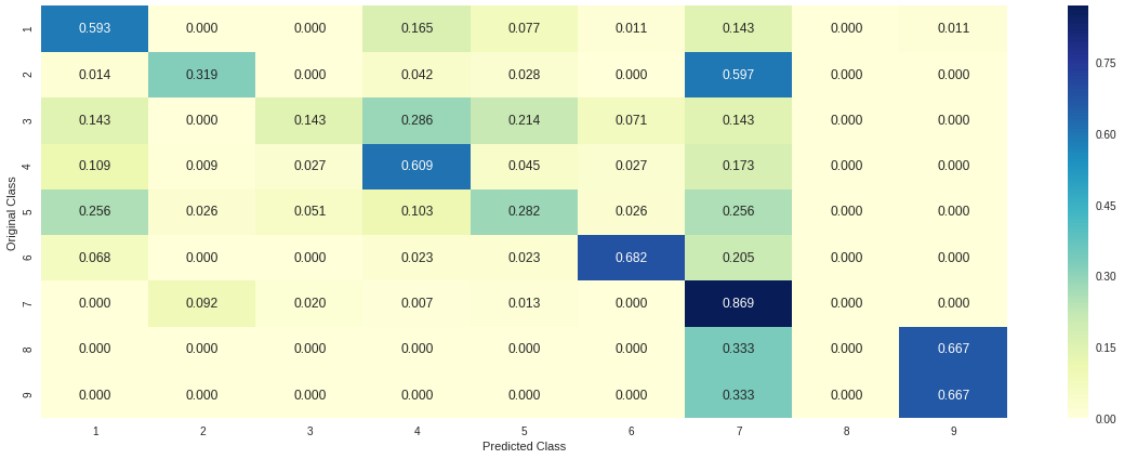
```
In [23]: #testing
         clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
         predict_and_plot_confusion_matrix(train_x_onehotCoding  , train_y, cv_x_onehotCoding
```

```
Log loss : 1.1442794503372584
Number of mis-classified points : 0.39097744360902253
-------------------- Confusion matrix --------------------
```

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



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

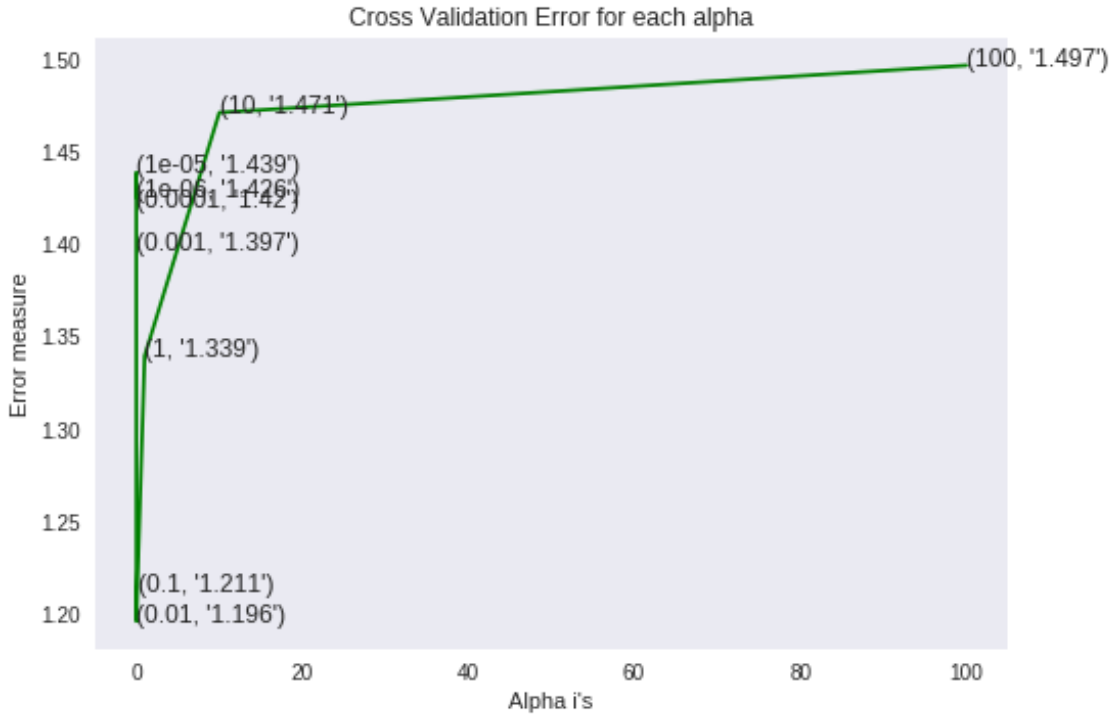


Without Class-Balancing

```
In [0]: alpha = [10 ** x for x in range(-6, 3)]
        cv_log_error_array = []
        for i in alpha:
            print("for alpha =", i)
            clf = 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
            # 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='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_lo
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding  )
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(test_x_onehotCoding  )
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
for alpha = 1e-06
Log Loss : 1.4258763958317966
for alpha = 1e-05
Log Loss : 1.4389348728297722
for alpha = 0.0001
Log Loss : 1.4200601192238345
for alpha = 0.001
Log Loss : 1.3966306255044216
for alpha = 0.01
Log Loss : 1.1956984251537213
for alpha = 0.1
Log Loss : 1.211324890462311
for alpha = 1
Log Loss : 1.3394525856859076
for alpha = 10
Log Loss : 1.4714680011093986
for alpha = 100
Log Loss : 1.4970678805864974
```



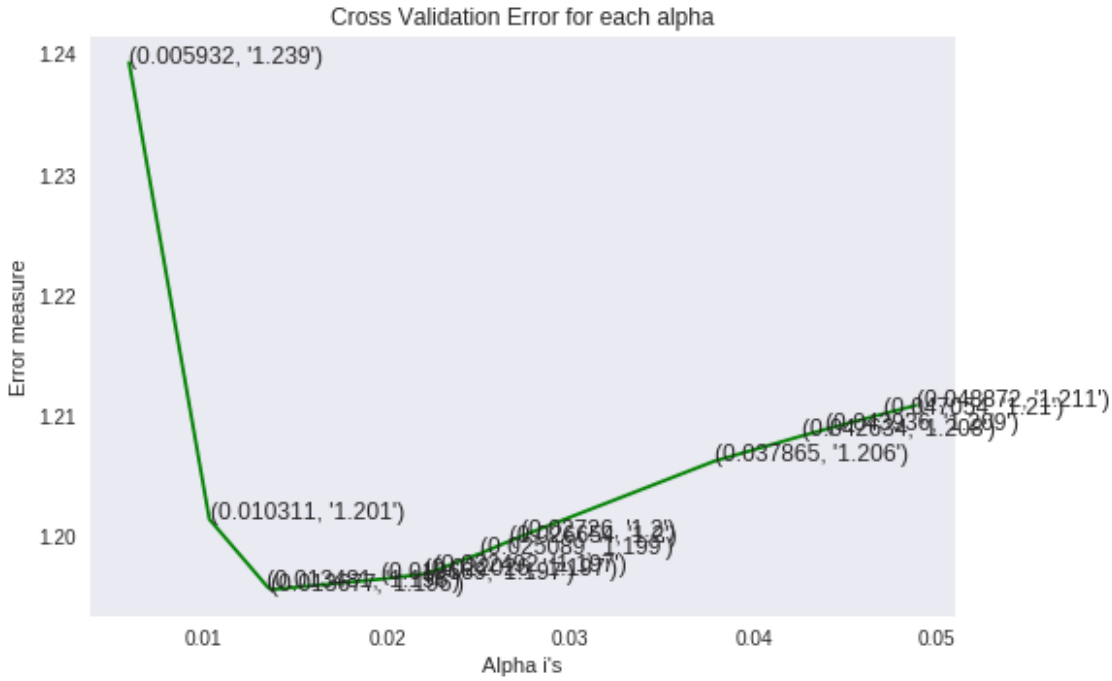

```
For values of best alpha =  0.01 The train log loss is: 0.8129006682318044
For values of best alpha =  0.01 The cross validation log loss is: 1.1956984251537213
For values of best alpha =  0.01 The test log loss is: 1.1458988499153788
```

```
In [24]: alpha = np.random.uniform(0.0005,0.05,15)
         alpha = np.round(alpha,6)
         alpha.sort()
         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=
             # 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='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_l
        predict_y = sig_clf.predict_proba(cv_x_onehotCoding  )
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(test_x_onehotCoding  )
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
```

```
for alpha = 0.005932
Log Loss : 1.2393024476347188
for alpha = 0.010311
Log Loss : 1.2014740404757616
for alpha = 0.013481
Log Loss : 1.1958714005495894
for alpha = 0.013677
Log Loss : 1.1956463872010756
for alpha = 0.019669
Log Loss : 1.1965150345879376
for alpha = 0.022026
Log Loss : 1.1969398416807377
for alpha = 0.022492
Log Loss : 1.1971594113553812
for alpha = 0.025089
Log Loss : 1.1986410817581856
for alpha = 0.026654
Log Loss : 1.1996432149828296
for alpha = 0.02726
Log Loss : 1.200032098589441
for alpha = 0.037865
Log Loss : 1.2062926860820418
for alpha = 0.042634
Log Loss : 1.2083802550691014
for alpha = 0.043936
Log Loss : 1.2089304134077374
for alpha = 0.047054
Log Loss : 1.2102007568719875
```

```
for alpha = 0.048872
Log Loss : 1.2109120378541194
```

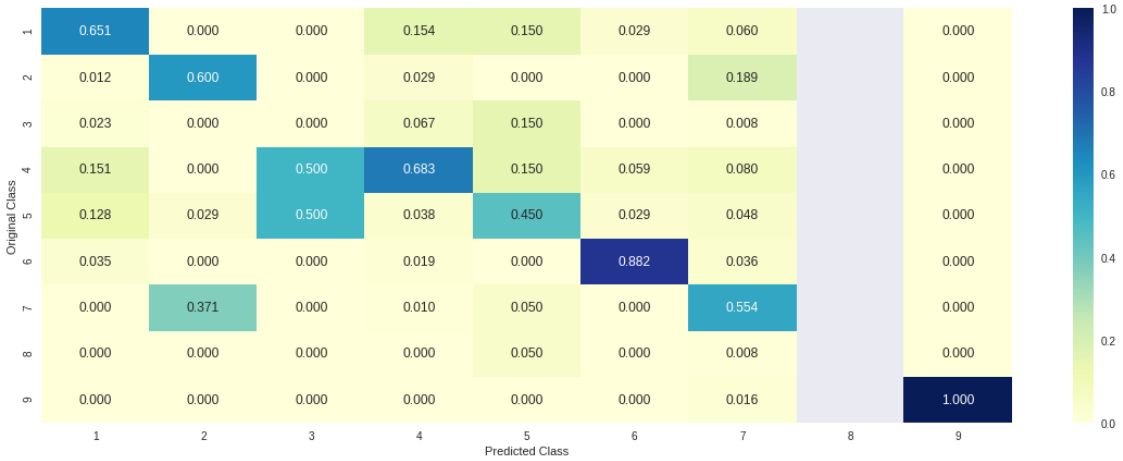



```
For values of best alpha =  0.013677 The train log loss is: 0.8493785540706176
For values of best alpha =  0.013677 The cross validation log loss is: 1.1956463872010756
For values of best alpha =  0.013677 The test log loss is: 1.1495628975008125
```

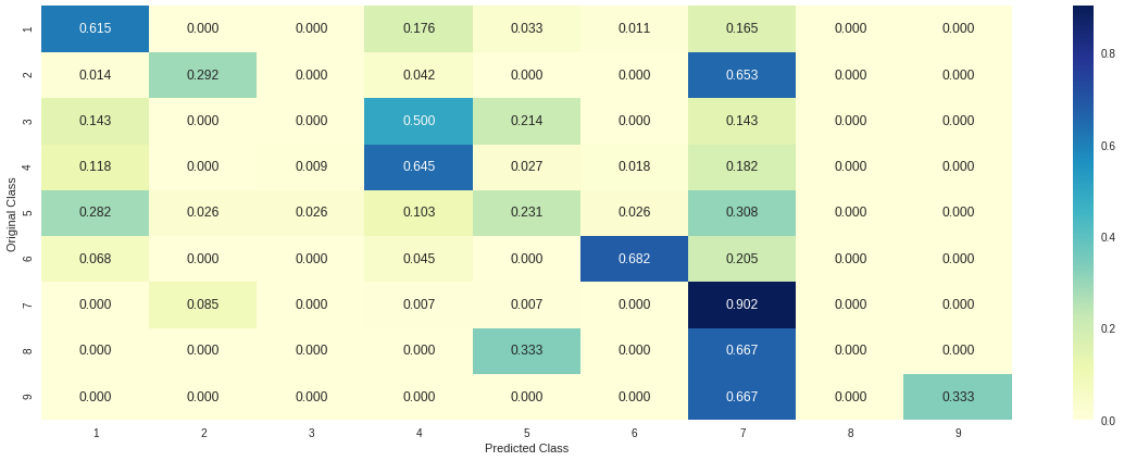
```
In [25]: #testing
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
         predict_and_plot_confusion_matrix(train_x_onehotCoding  , train_y, cv_x_onehotCoding
```

```
Log loss : 1.1956463872010756
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```

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



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



3.1.1 Feature Importance

```
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(train_text_features  )
                    word = temp1[v-(fea1_len+fea2_len)]
                    l = word.split()
                    if len(l) ==2:
                        if l[0] in text.split() and l[1] in text.split():
                            yes_no = True
                        else:
                            yes_no = False
                    else:
                        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(wor

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

```
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 = 52
        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_onehotCod
        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: [[0.3024 0.0926 0.0162 0.3745 0.0454 0.0336 0.1221 0.0066 0.0065
Actual Class : 5
--------------------------------------------------
22 Text feature [level no] present in test data point [True]
24 Text feature [repression thus] present in test data point [True]
29 Text feature [specifically or] present in test data point [True]
59 Text feature [cause such] present in test data point [True]
63 Text feature [at between] present in test data point [True]
69 Text feature [mutant 48] present in test data point [True]
98 Text feature [expression displayed] present in test data point [True]
134 Text feature [thus p53] present in test data point [True]
141 Text feature [family from] present in test data point [True]
151 Text feature [likely other] present in test data point [True]
179 Text feature [generated or] present in test data point [True]
183 Text feature [boundary in] present in test data point [True]
191 Text feature [was conserved] present in test data point [True]
193 Text feature [acetylation to] present in test data point [True]
230 Text feature [observed partial] present in test data point [True]
```

```
234 Text feature [including differential] present in test data point [True]
236 Text feature [its general] present in test data point [True]
239 Text feature [activity involved] present in test data point [True]
241 Text feature [but bound] present in test data point [True]
243 Text feature [family does] present in test data point [True]
253 Text feature [the mentioned] present in test data point [True]
254 Text feature [other base] present in test data point [True]
255 Text feature [residues analyzed] present in test data point [True]
260 Text feature [cancer suppressor] present in test data point [True]
266 Text feature [suppressor function] present in test data point [True]
278 Text feature [observed confirming] present in test data point [True]
284 Text feature [functional difference] present in test data point [True]
286 Text feature [indicate cancer] present in test data point [True]
292 Text feature [some mutation] present in test data point [True]
295 Text feature [some transactivation] present in test data point [True]
297 Text feature [date this] present in test data point [True]
302 Text feature [including removal] present in test data point [True]
313 Text feature [has even] present in test data point [True]
320 Text feature [acetylation which] present in test data point [True]
335 Text feature [when variants] present in test data point [True]
343 Text feature [not protein] present in test data point [True]
346 Text feature [not p53] present in test data point [True]
352 Text feature [and protecting] present in test data point [True]
408 Text feature [mammalian base] present in test data point [True]
428 Text feature [domains specifically] present in test data point [True]
442 Text feature [transcriptional crystal] present in test data point [True]
482 Text feature [efficiently second] present in test data point [True]
496 Text feature [which structural] present in test data point [True]
Out of the top  500  features  43 are present in query point
```

```
In [0]: # from tabulate import tabulate
        clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
        clf.fit(train_x_onehotCoding  ,train_y)
        test_point_index = 10
        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_onehotCo
        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: [[0.053 0.107 0.01  0.047 0.025 0.173 0.575 0.007 0.004]]
Actual Class : 7
--------------------------------------------------
```

```
47 Text feature [downstream signaling] present in test data point [True]
59 Text feature [mutations increased] present in test data point [True]
62 Text feature [normal thyroid] present in test data point [True]
72 Text feature [activation pathway] present in test data point [True]
86 Text feature [constitutively activate] present in test data point [True]
107 Text feature [and ectodermal] present in test data point [True]
113 Text feature [or previously] present in test data point [True]
120 Text feature [identify oncogenes] present in test data point [True]
129 Text feature [analysis then] present in test data point [True]
167 Text feature [the transformation] present in test data point [True]
187 Text feature [constitutively] present in test data point [True]
202 Text feature [levels either] present in test data point [True]
257 Text feature [data occur] present in test data point [True]
286 Text feature [transformation whereas] present in test data point [True]
289 Text feature [samples clinical] present in test data point [True]
294 Text feature [transformation was] present in test data point [True]
295 Text feature [complex showing] present in test data point [True]
303 Text feature [type genes] present in test data point [True]
308 Text feature [human thyroid] present in test data point [True]
316 Text feature [thyroid] present in test data point [True]
318 Text feature [between 100] present in test data point [True]
320 Text feature [ectodermal anomalies] present in test data point [True]
334 Text feature [proteins 13] present in test data point [True]
339 Text feature [samples approximately] present in test data point [True]
341 Text feature [encode in] present in test data point [True]
343 Text feature [had rearrangements] present in test data point [True]
351 Text feature [both inhibitors] present in test data point [True]
378 Text feature [oncogenes we] present in test data point [True]
387 Text feature [similar activation] present in test data point [True]
398 Text feature [one more] present in test data point [True]
407 Text feature [as commonly] present in test data point [True]
411 Text feature [downstream kinase] present in test data point [True]
422 Text feature [two that] present in test data point [True]
427 Text feature [only cells] present in test data point [True]
432 Text feature [patients four] present in test data point [True]
441 Text feature [had relative] present in test data point [True]
468 Text feature [phosphorylation provide] present in test data point [True]
476 Text feature [other constitutively] present in test data point [True]
485 Text feature [previously seen] present in test data point [True]
495 Text feature [more evidence] present in test data point [True]
Out of the top  500  features  40 are present in query point
```

---

4 Conclusion

Best results obtained when using uni-bi grams gives a test loss of 1.145 with Logistic Regression without class balancing.