CancerDiagnosis-Assignment-4

July 24, 2018

1 Importing the libraries

In [2]:
```
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)
```

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...
```

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
```

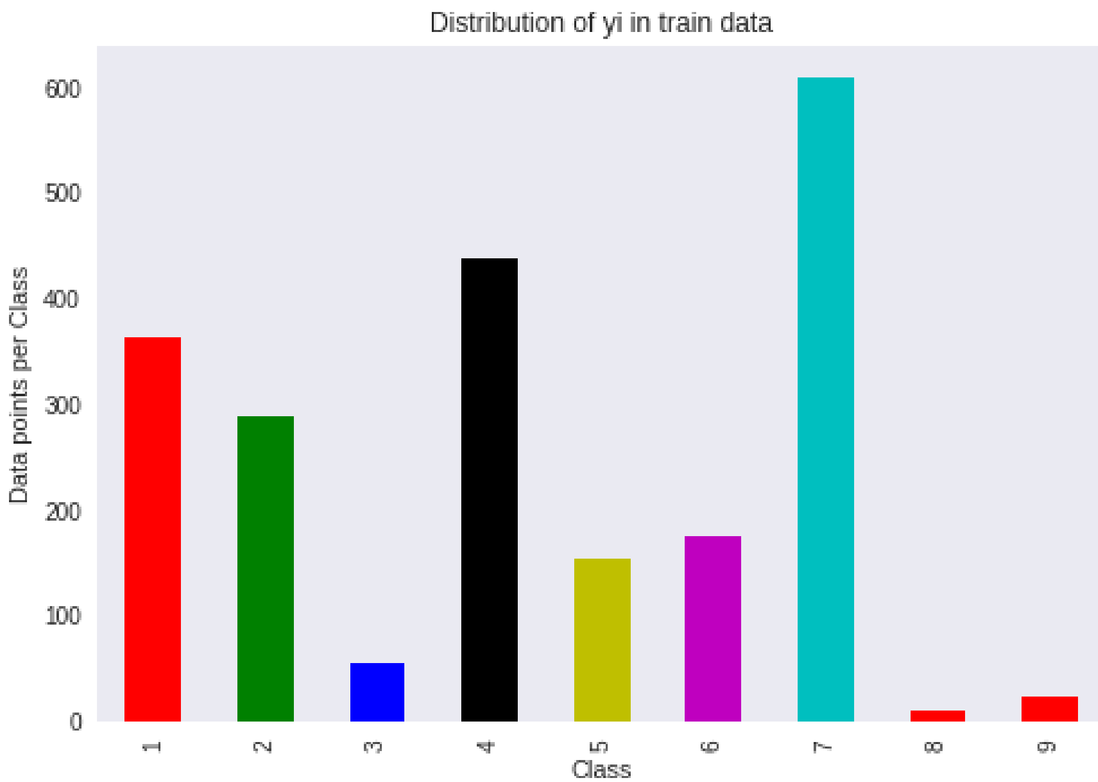
In [12]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

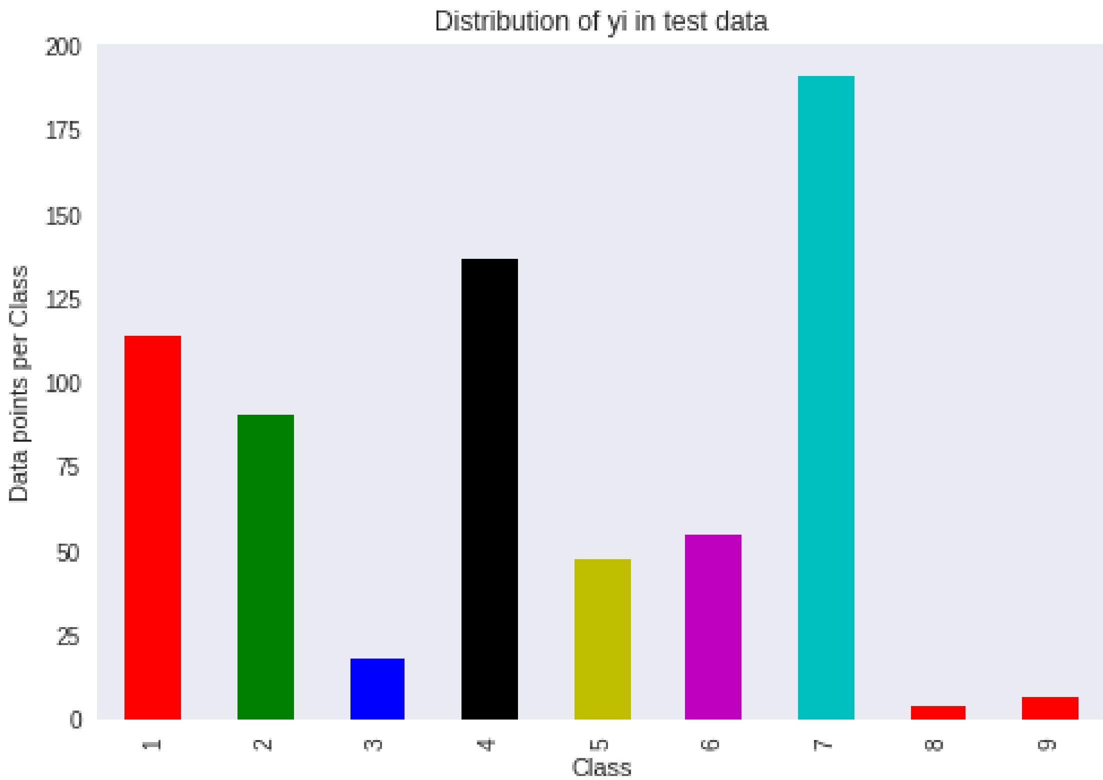
Distribution of y_i's in Train, Test and Cross Validation datasets

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']
```

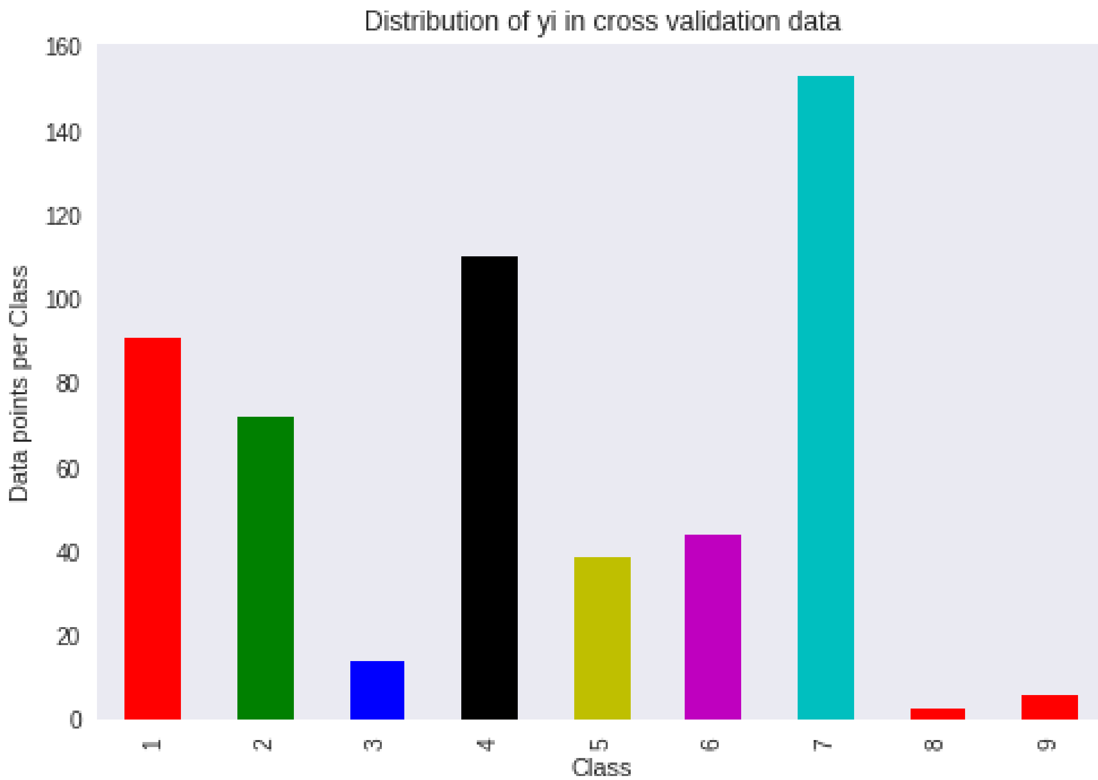
```
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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 Feature Engineering

2.1 Tf-idf Vectorization

```
In [0]: 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'])
```

```
In [0]: variation_vectorizer = TfidfVectorizer()
        train_variation_feature_tfidfCoding = variation_vectorizer.fit_transform(train_df['Var
```

```
test_variation_feature_tfidfCoding = variation_vectorizer.transform(test_df['Variation
cv_variation_feature_tfidfCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
In [16]: # building a TfidfVectorizer with all the words that occured minimum 3 times in train
         text_vectorizer = TfidfVectorizer(min_df=3,ngram_range=(1,2))
         train_text_feature_tfidfCoding = text_vectorizer.fit_transform(train_df['TEXT'].values
         # 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_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))

         # we use the same vectorizer that was trained on train data
         test_text_feature_tfidfCoding = text_vectorizer.transform(test_df['TEXT'].values.asty

         # we use the same vectorizer that was trained on train data
         cv_text_feature_tfidfCoding = text_vectorizer.transform(cv_df['TEXT'].values.astype('
```

```
Total number of unique words in train data : 689702
```

2.1.1 Stacking Features

```
In [0]: train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_fe
        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, 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()
```

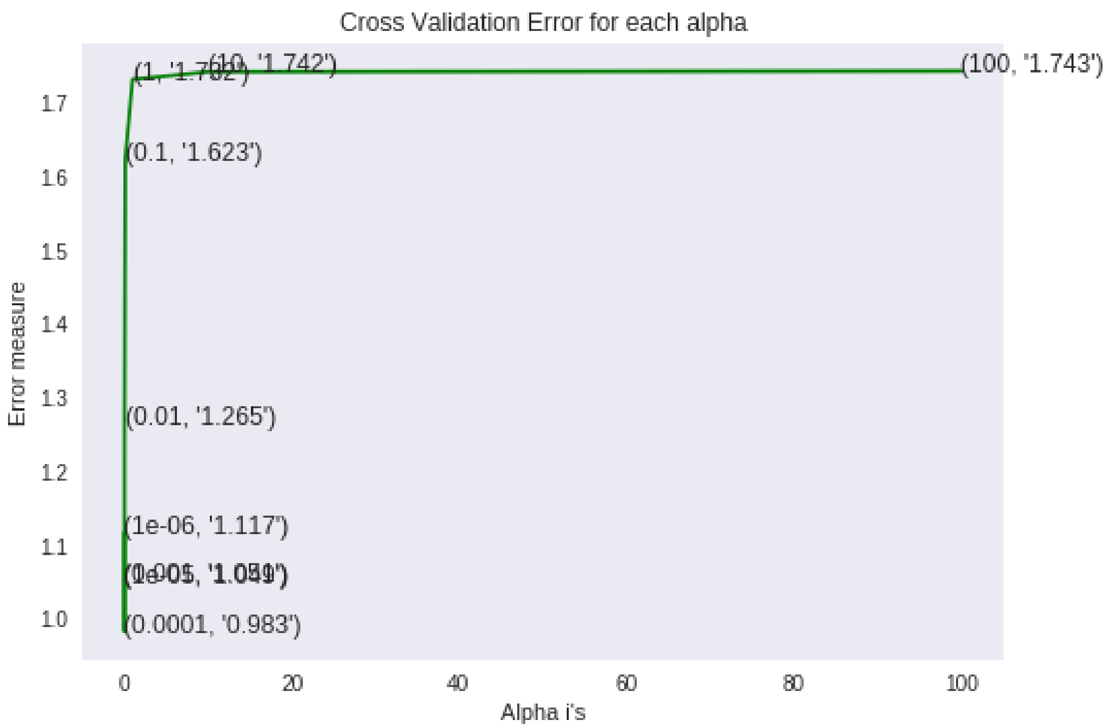
```
In [0]: def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
            clf.fit(train_x, train_y)
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
            sig_clf.fit(train_x, train_y)
            pred_y = sig_clf.predict(test_x)

            # for calculating log_loss we willl provide the array of probabilities belongs to e
            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 Tf-idf with class Balancing

```
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_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', 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 = 1e-06
Log Loss : 1.1173583310566364
for alpha = 1e-05
Log Loss : 1.0489317387661579
for alpha = 0.0001
Log Loss : 0.9829321078049326
for alpha = 0.001
Log Loss : 1.0509021734916737
for alpha = 0.01
Log Loss : 1.2653125875132296
for alpha = 0.1
Log Loss : 1.6230437860034812
for alpha = 1
Log Loss : 1.731839794164275
for alpha = 10
Log Loss : 1.741956713390952
for alpha = 100
Log Loss : 1.7430117264073257
```



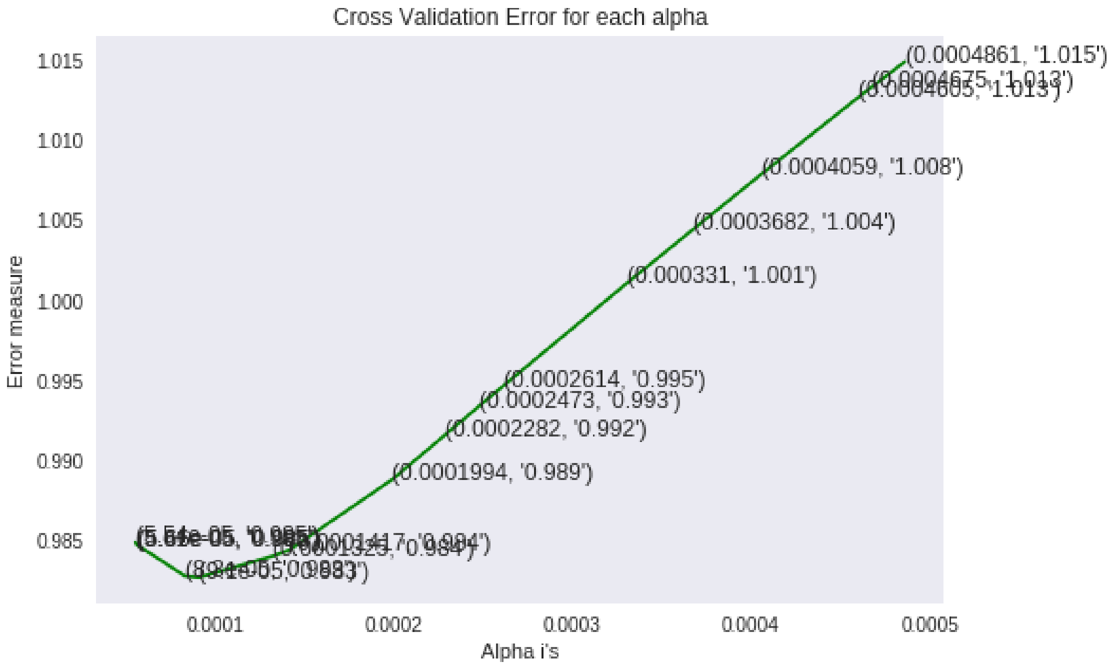

```
For values of best alpha =  0.0001 The train log loss is: 0.42228639710437893
For values of best alpha =  0.0001 The cross validation log loss is: 0.9829321078049326
```

```
For values of best alpha =  0.0001 The test log loss is: 0.980026539861125
```

```
In [21]: #alpha = [10 ** x for x in range(-6, 3)]
         alpha = np.random.uniform(0.00005,0.0005,17)
         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=1
             # 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 = 5.54e-05
Log Loss : 0.984913664225747
for alpha = 5.61e-05
Log Loss : 0.9847374783231352
for alpha = 5.66e-05
```

```
Log Loss : 0.9847245376850747
for alpha = 8.3e-05
Log Loss : 0.982811236304561
for alpha = 9.1e-05
Log Loss : 0.9827183610043397
for alpha = 0.0001325
Log Loss : 0.9840483684707169
for alpha = 0.0001417
Log Loss : 0.9844243233412948
for alpha = 0.0001994
Log Loss : 0.9888373829710956
for alpha = 0.0002282
Log Loss : 0.9915493495412293
for alpha = 0.0002473
Log Loss : 0.9933503565433187
for alpha = 0.0002614
Log Loss : 0.9946703098643686
for alpha = 0.000331
Log Loss : 1.0011182278799904
for alpha = 0.0003682
Log Loss : 1.0044891085974934
for alpha = 0.0004059
Log Loss : 1.0078903445189553
for alpha = 0.0004605
Log Loss : 1.0127135409116708
for alpha = 0.0004675
Log Loss : 1.0133165023982373
for alpha = 0.0004861
Log Loss : 1.0149013415345334
```




```
For values of best alpha =  9.1e-05 The train log loss is: 0.417878443414042 57
For values of best alpha =  9.1e-05 The cross validation log loss is: 0.9827183610043397
For values of best alpha =  9.1e-05 The test log loss is: 0.9797665201082526
```

```
In [22]: 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 : 0.9827183610043397
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```

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


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

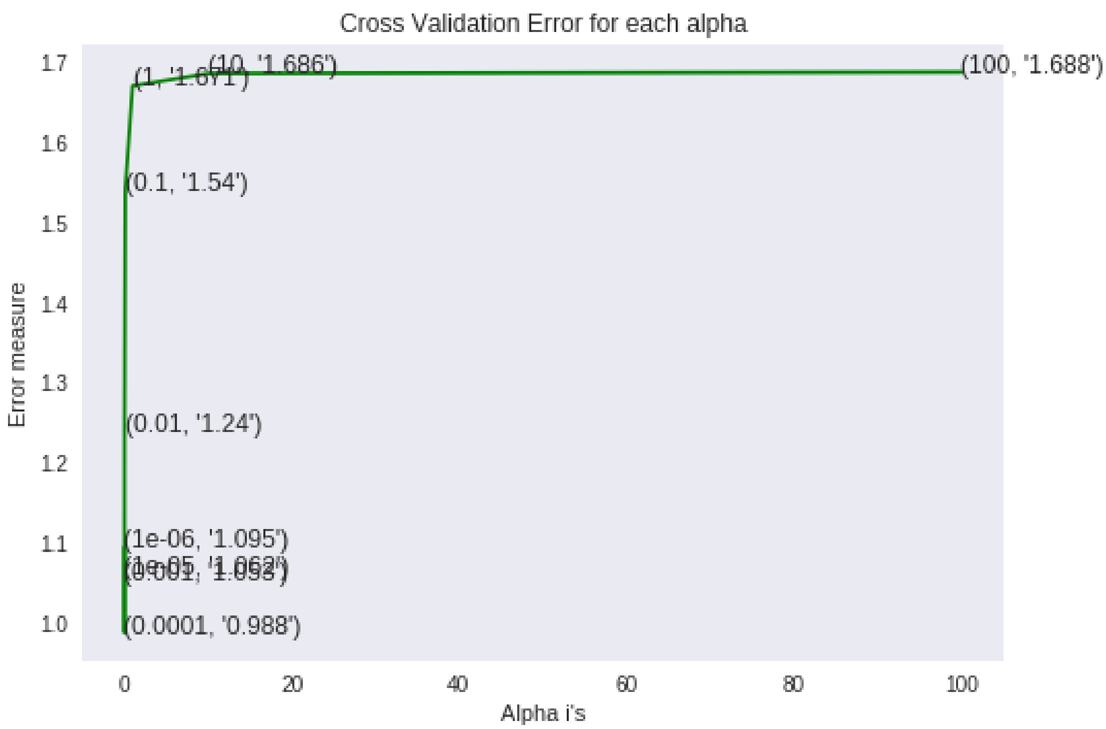
3.2 Without Class-Balancing

In [24]:
```python
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_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='log', 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_
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_l
```

```
for alpha = 1e-06
Log Loss : 1.0950210266890523
for alpha = 1e-05
Log Loss : 1.062283307296617
for alpha = 0.0001
Log Loss : 0.9880175040395316
for alpha = 0.001
Log Loss : 1.0526897700956737
for alpha = 0.01
Log Loss : 1.2395918102488723
for alpha = 0.1
Log Loss : 1.540137685642479
for alpha = 1
```

```
Log Loss : 1.6705673707068973
for alpha = 10
Log Loss : 1.6861318381653372
for alpha = 100
Log Loss : 1.6878400669728781
```



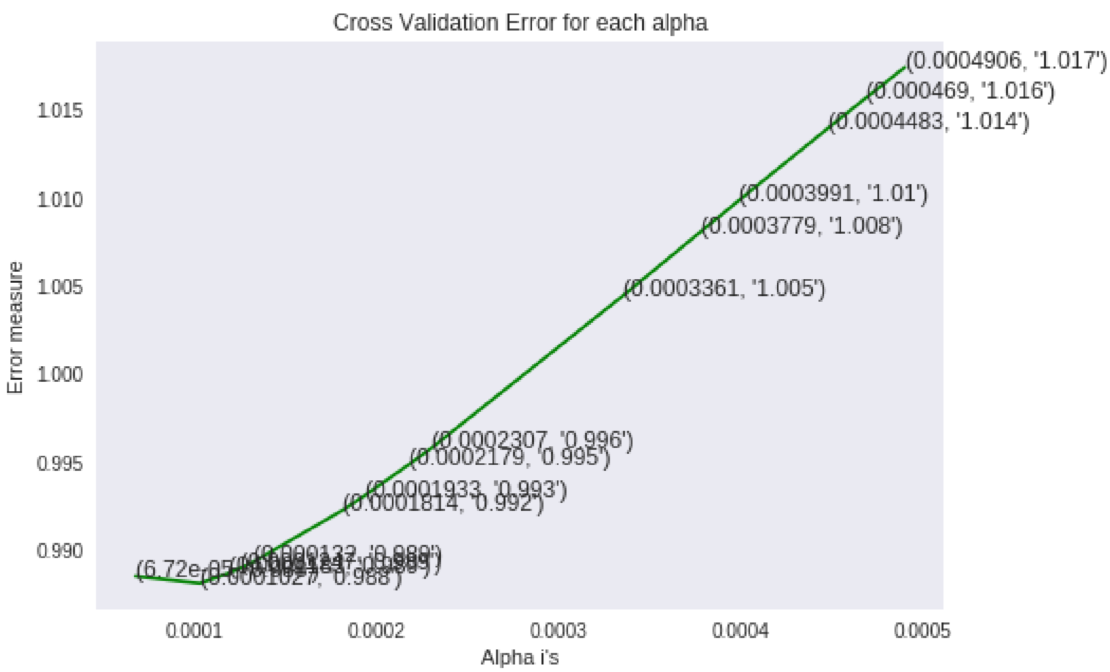

```
For values of best alpha =  0.0001 The train log loss is: 0.41185946149945185
For values of best alpha =  0.0001 The cross validation log loss is: 0.9880175040395316
For values of best alpha =  0.0001 The test log loss is: 0.9788678297825318
```

In [25]:
```
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='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 = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', 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_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 = 6.72e-05
Log Loss : 0.9884967444643222
for alpha = 0.0001027
Log Loss : 0.9880923557706298
for alpha = 0.0001183
Log Loss : 0.9886595905916418
for alpha = 0.0001247
Log Loss : 0.988944123897926
for alpha = 0.000132
Log Loss : 0.98929765348536
for alpha = 0.0001814
Log Loss : 0.9922517910824706
for alpha = 0.0001933
Log Loss : 0.9930695482727672
for alpha = 0.0002179
Log Loss : 0.9948560299924325
for alpha = 0.0002307
Log Loss : 0.9958290737523492
for alpha = 0.0003361
Log Loss : 1.00450012837579
for alpha = 0.0003779
```

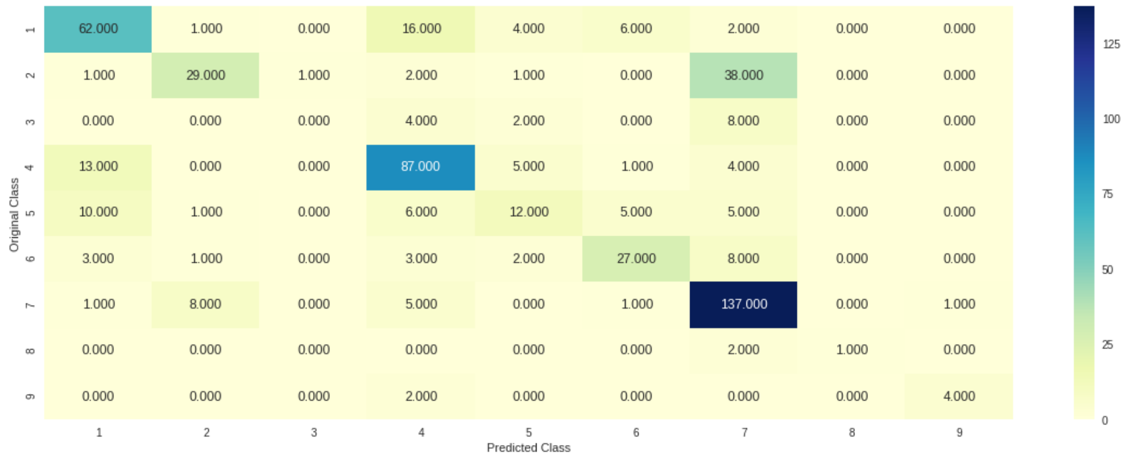
```
Log Loss : 1.0080594522057722
for alpha = 0.0003991
Log Loss : 1.0098577201645929
for alpha = 0.0004483
Log Loss : 1.0139819967752568
for alpha = 0.000469
Log Loss : 1.0156900227740726
for alpha = 0.0004906
Log Loss : 1.0174523731056162
```



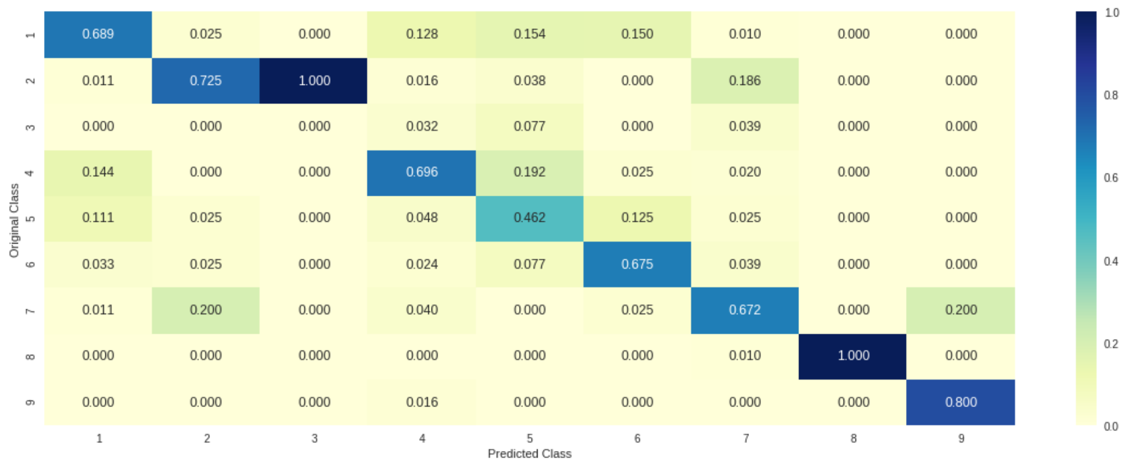

```
For values of best alpha =  0.0001027 The train log loss is: 0.4130564402532021
For values of best alpha =  0.0001027 The cross validation log loss is: 0.9880923557706298
For values of best alpha =  0.0001027 The test log loss is: 0.9788389791659189
```

```
In [26]: 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_
```

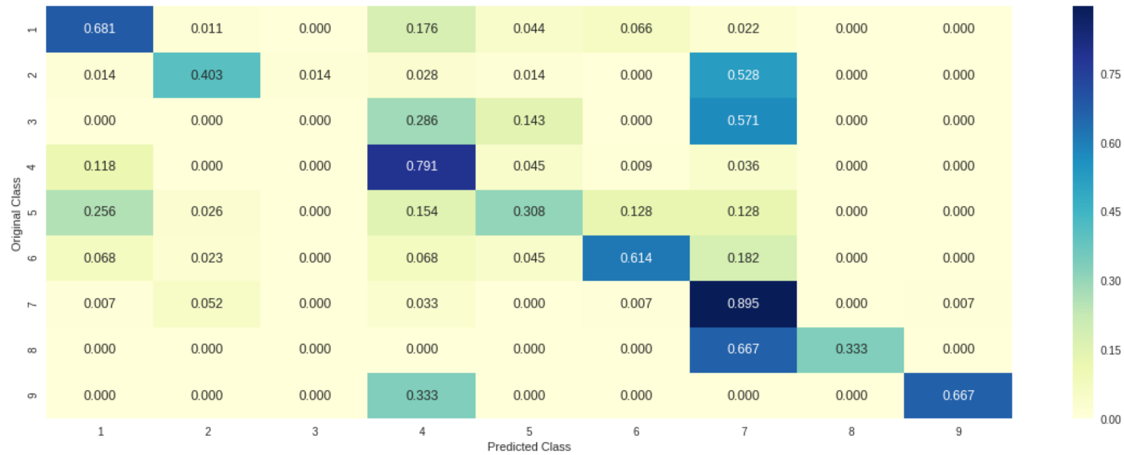
```
Log loss : 0.9880923557706298
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```



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



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



4 Conclusion

Best Results are obtained using tf-idf vectorization technique with class balancing features. The test loss obtained using the same is **0.97** which is about **64**% improvement over a random model