market-segmentation-ml-models

October 11, 2023

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```python
# Import all required Libraries:

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 import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
```

```
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
import six
import sys
sys.modules['sklearn.externals.six'] = six
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

[ ]:
```
df = pd.read_csv("/content/drive/MyDrive/out.csv")
```

[ ]:
```
df.head()
```

[ ]:
```
   yummy  convenient  spicy  fattening  greasy  fast  cheap  tasty  expensive  \
0      0           1      0          1       0     1      1      0          1
1      1           1      0          1       1     1      1      1          1
2      0           1      1          1       1     1      0      1          1
3      1           1      0          1       1     1      1      1          0
4      0           1      0          1       1     1      1      0          0

   healthy  disgusting  Like  Age  VisitFrequency  Gender  Cluster
0        0           0    -3   61               2       0        3
1        0           0     2   51               2       0        0
2        1           0     1   62               2       0        0
3        0           1     4   69               4       0        2
4        1           0     2   49               3       1        2
```

[ ]:
```
df.shape
```

[ ]:
```
(1453, 16)
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['yummy', 'convenient', 'spicy', 'fattening', 'greasy', 'fast', 'cheap',
       'tasty', 'expensive', 'healthy', 'disgusting', 'Like', 'Age',
       'VisitFrequency', 'Gender', 'Cluster'],
      dtype='object')
```

[ ]:
```
# columns to keep:
data= df[['yummy', 'convenient', 'spicy', 'fattening', 'greasy', 'fast',␣
 ↪'cheap',
       'tasty', 'expensive', 'healthy', 'disgusting', 'Age', 'Gender',␣
 ↪'Cluster']].rename({'Cluster':'label'},axis=1)
```

[ ]:
```
data.head(2)
```

```
[ ]:    yummy  convenient  spicy  fattening  greasy  fast  cheap  tasty  expensive  \
     0      0           1      0          1       0     1      1      0          1
     1      1           1      0          1       1     1      1      1          1

        healthy  disgusting  Age  Gender  label
     0        0           0   61       0      3
     1        0           0   51       0      0
```

```
[ ]: X = data.iloc[:, data.columns != 'label']
     y = data.iloc[:, data.columns == 'label']
```

0.1 Train ,Test and Cross-Validation Dataset Construction

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

```
[ ]: 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: 929
Number of data points in test data: 291
Number of data points in cross validation data: 233
```

```
[ ]: test_df.head(2)
```

```
[ ]:       yummy  convenient  spicy  fattening  greasy  fast  cheap  tasty  \
     140       0           1      0          1       0     1      1      1
     1349      1           1      0          1       0     1      0      1

           expensive  healthy  disgusting  Age  Gender
     140           0        0           0   42       1
     1349          1        1           0   31       1
```

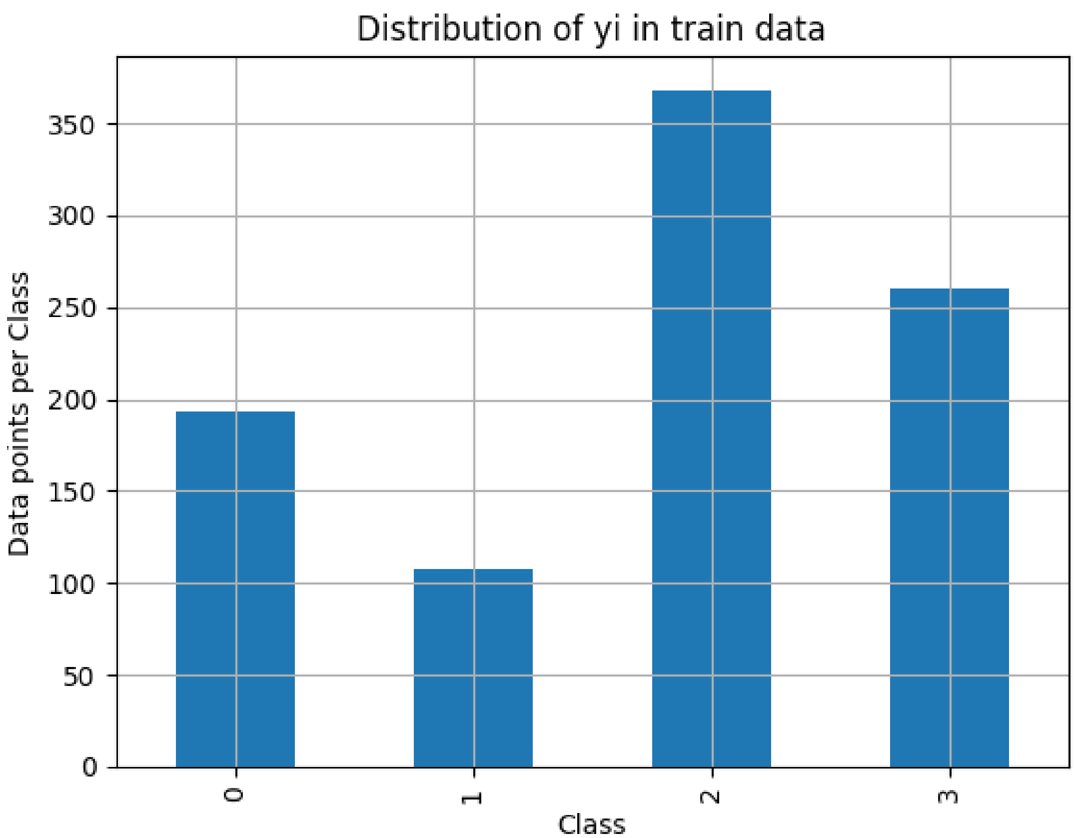
```
[ ]: y_test.head(2)
```

```
[ ]:       label
     140       2
     1349      0
```

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

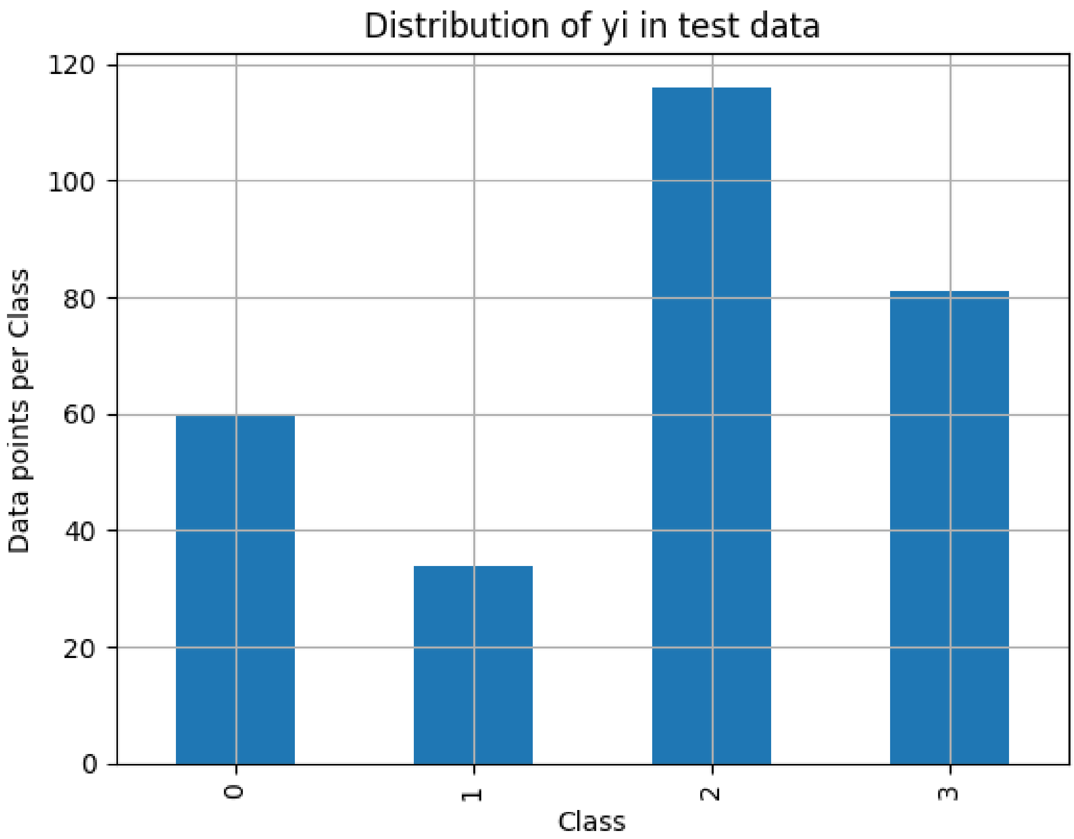
[ ]:
```
# it returns a dict, keys as class labels and values as the number of data␣
↪points in that class
train_class_distribution = y_train['label'].value_counts().sort_index()
test_class_distribution = y_test['label'].value_counts().sort_index()
cv_class_distribution = y_cv ['label'].value_counts().sort_index()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.
↪argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing␣
↪order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.
↪values[i], '(', np.round((train_class_distribution.values[i]/train_df.
↪shape[0]*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.
↪argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing␣
↪order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.
↪values[i], '(', np.round((test_class_distribution.values[i]/test_df.
↪shape[0]*100), 3), '%)')

print('-'*80)
my_colors = 'rgbkymc'
```

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.
↪argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing␣
↪order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.
↪values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.
↪shape[0]*100), 3), '%)')
```

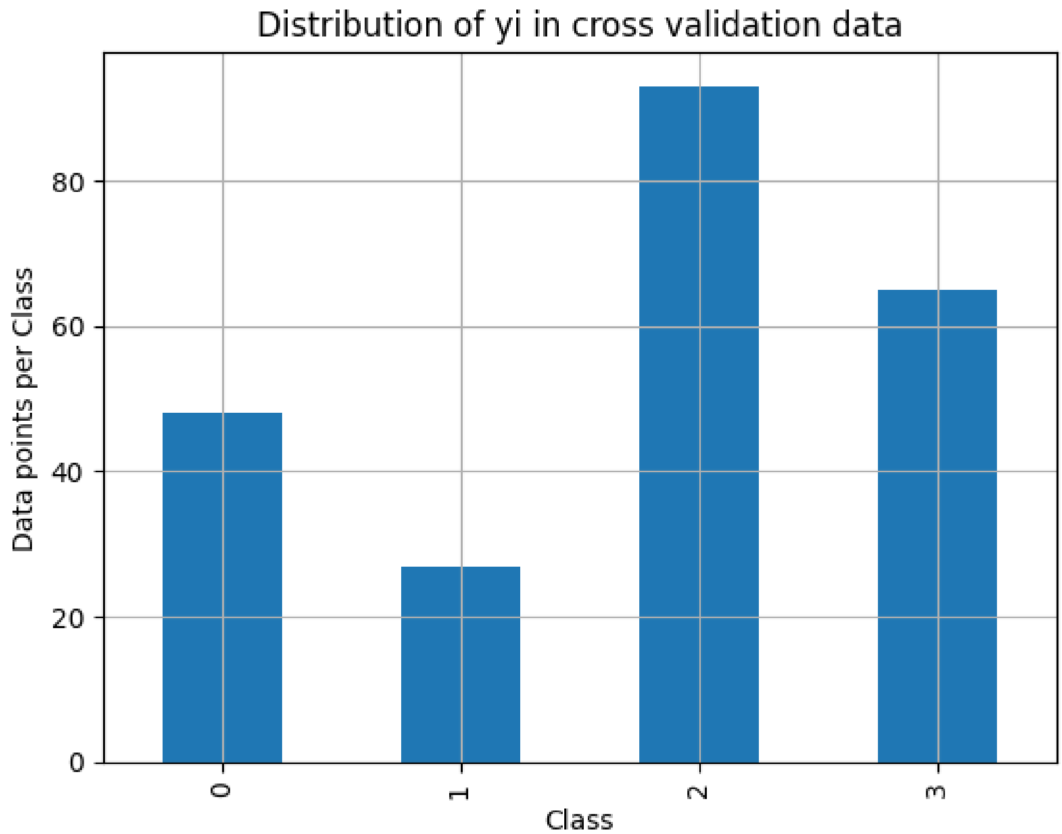



```
Number of data points in class 3 : 368 ( 39.612 %)
Number of data points in class 4 : 260 ( 27.987 %)
Number of data points in class 1 : 193 ( 20.775 %)
```

```
Number of data points in class 2 : 108 ( 11.625 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 3 : 116 ( 39.863 %)
Number of data points in class 4 : 81 ( 27.835 %)
Number of data points in class 1 : 60 ( 20.619 %)
Number of data points in class 2 : 34 ( 11.684 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 3 : 93 ( 39.914 %)
Number of data points in class 4 : 65 ( 27.897 %)
Number of data points in class 1 : 48 ( 20.601 %)
Number of data points in class 2 : 27 ( 11.588 %)
```

[ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_df= scaler.fit_transform(train_df)
train_df = pd.DataFrame(train_df)
test_df = scaler.transform(test_df)
test_df = pd.DataFrame(test_df)
cv_df = scaler.transform(cv_df)
cv_df = pd.DataFrame(cv_df)
```

Prediction using a 'Random' Model

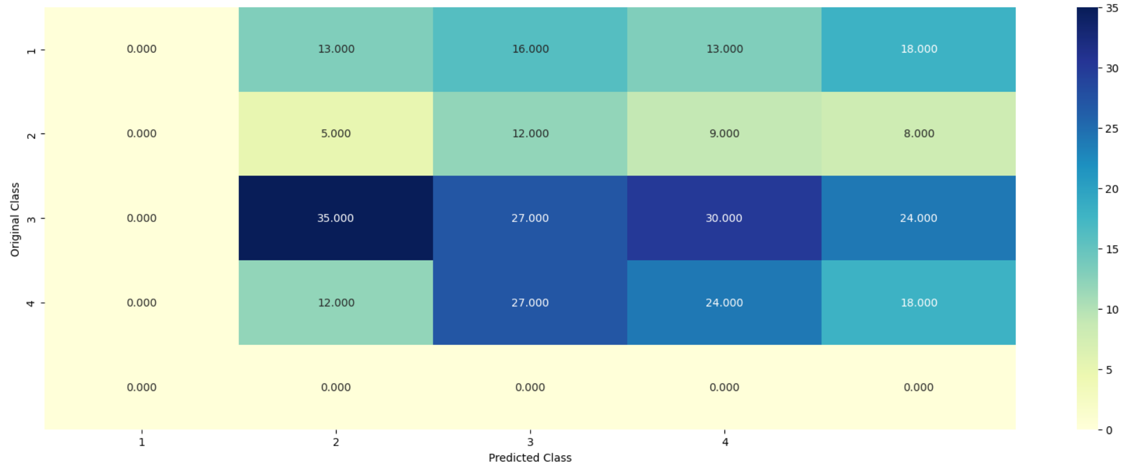
[ ]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
```

```
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i␣
↪are predicted class j

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

    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to␣
↪rows in two diamensional array
    # 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 two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                       [3/4, 4/6]]

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

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels,␣
↪yticklabels=labels)
```

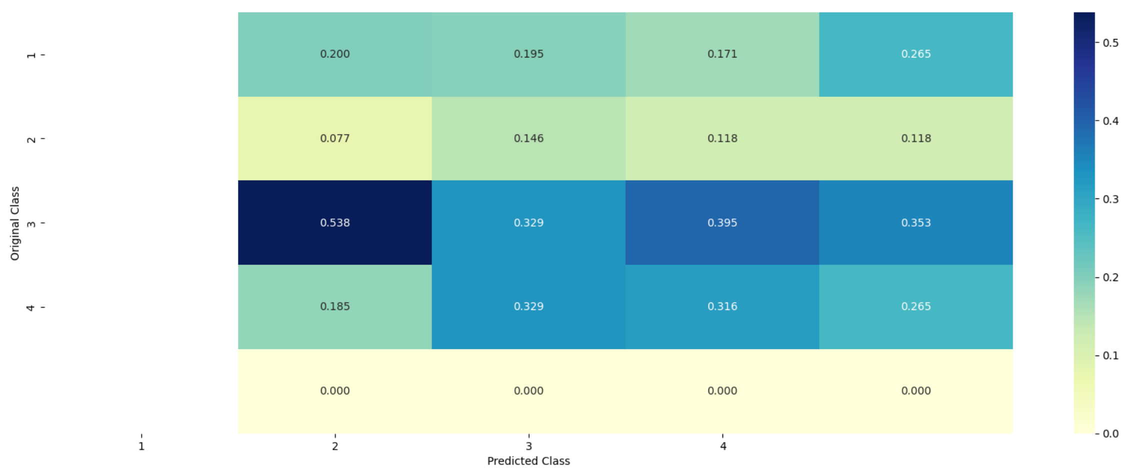
```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels,␣
↪yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

[ ]:
```
# we need to generate 5 numbers and the sum of numbers should be 1
# one solution is to genarate 5 numbers and divide each of the numbers by their␣
↪sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,4))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,4)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random␣
↪Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,4))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,4)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random␣
↪Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

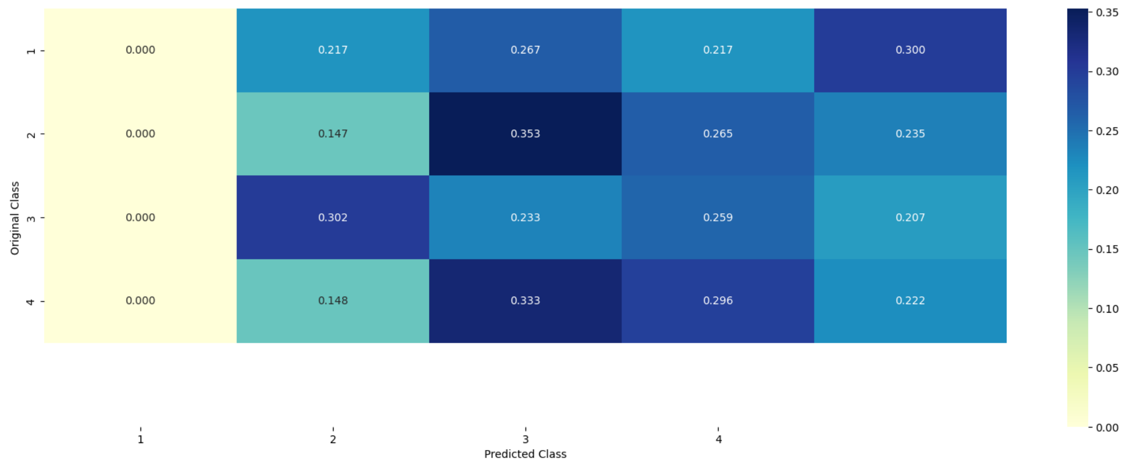
```
Log loss on Cross Validation Data using Random Model 1.6801866526522178
Log loss on Test Data using Random Model 1.611141956382239
-------------------- Confusion matrix --------------------
```



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



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



Machine Learning Models

[ ]:
```python
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities␣
 ↪belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y-␣
 ↪test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

[ ]:
```python
import pickle
pickle.dump(sig_clf, open('/content/drive/MyDrive/final_prediction.pickle',␣
 ↪'wb'))
pickle.dump(scaler, open('/content/drive/MyDrive/scaler.pickle', 'wb'))def␣
 ↪report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

```
  File "<ipython-input-48-bf421f3cc827>", line 3
    pickle.dump(scaler, open('/content/drive/MyDrive/scaler.pickle', 'wb'))def␣
 ↪report_log_loss(train_x, train_y, test_x, test_y,  clf):
                                                                              ^

SyntaxError: invalid syntax
```

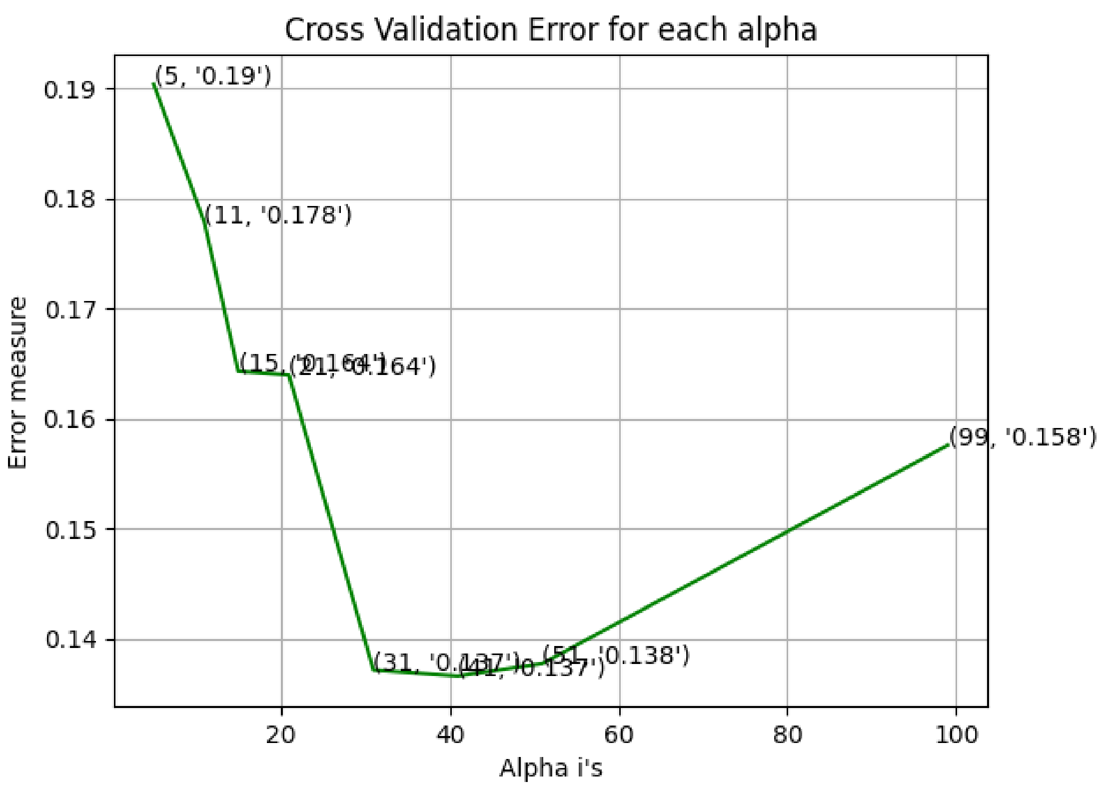
K Nearest Neighbour Classification

Hyper parameter tuning

[ ]:
```
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_df, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df, y_train)
    sig_clf_probs = sig_clf.predict_proba(cv_df)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use␣
 ↪log-probability estimates
    print("Log Loss :",log_loss(y_cv, 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_df, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df, y_train)

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

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

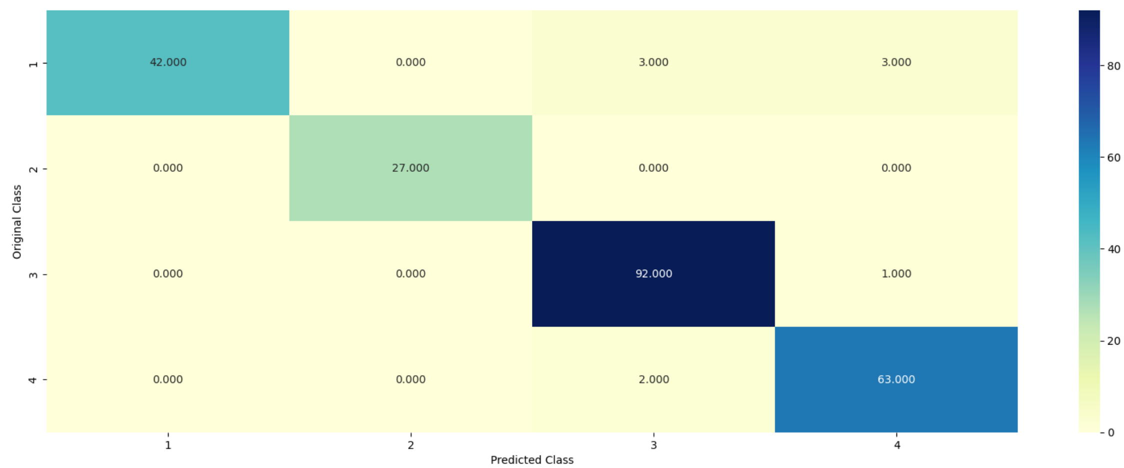
```
for alpha = 5
Log Loss : 0.1903307935393392
for alpha = 11
Log Loss : 0.1777915041878316
for alpha = 15
Log Loss : 0.16426317498928972
for alpha = 21
Log Loss : 0.16391829819291193
for alpha = 31
Log Loss : 0.1371292146305823
for alpha = 41
Log Loss : 0.13658723817517318
for alpha = 51
Log Loss : 0.13771713580884173
for alpha = 99
Log Loss : 0.1575477268166864
```



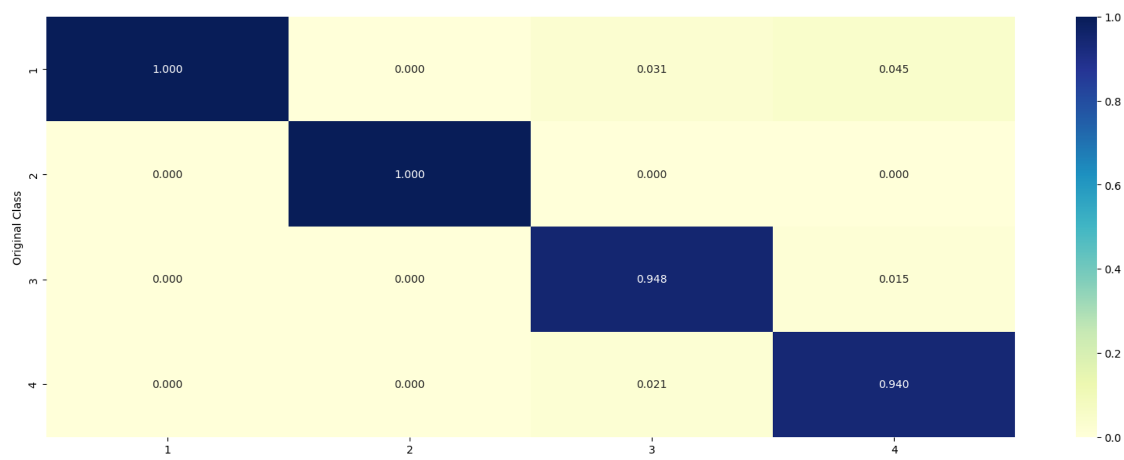
```
For values of best alpha =  41 The train log loss is: 0.17882548673489387
For values of best alpha =  41 The cross validation log loss is:
0.13658723817517318
For values of best alpha =  41 The test log loss is: 0.16799609578880664
```

[ ]:
```python
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_df.values, y_train.values, cv_df.
 ↪values, y_cv.values, clf)
```

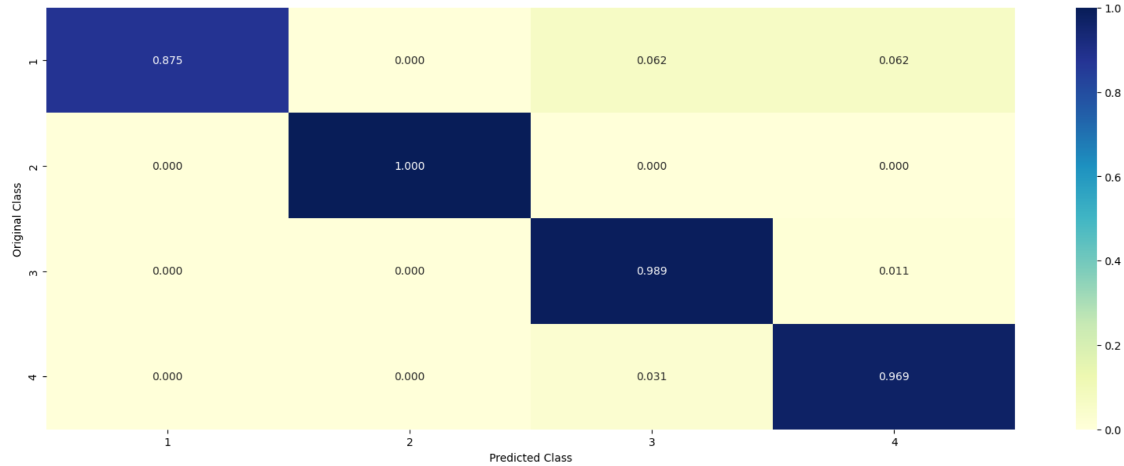
```
Log loss : 0.13658723817517318
Number of mis-classified points : 163.81115879828326
-------------------- Confusion matrix --------------------
```




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



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



Logistic Regression

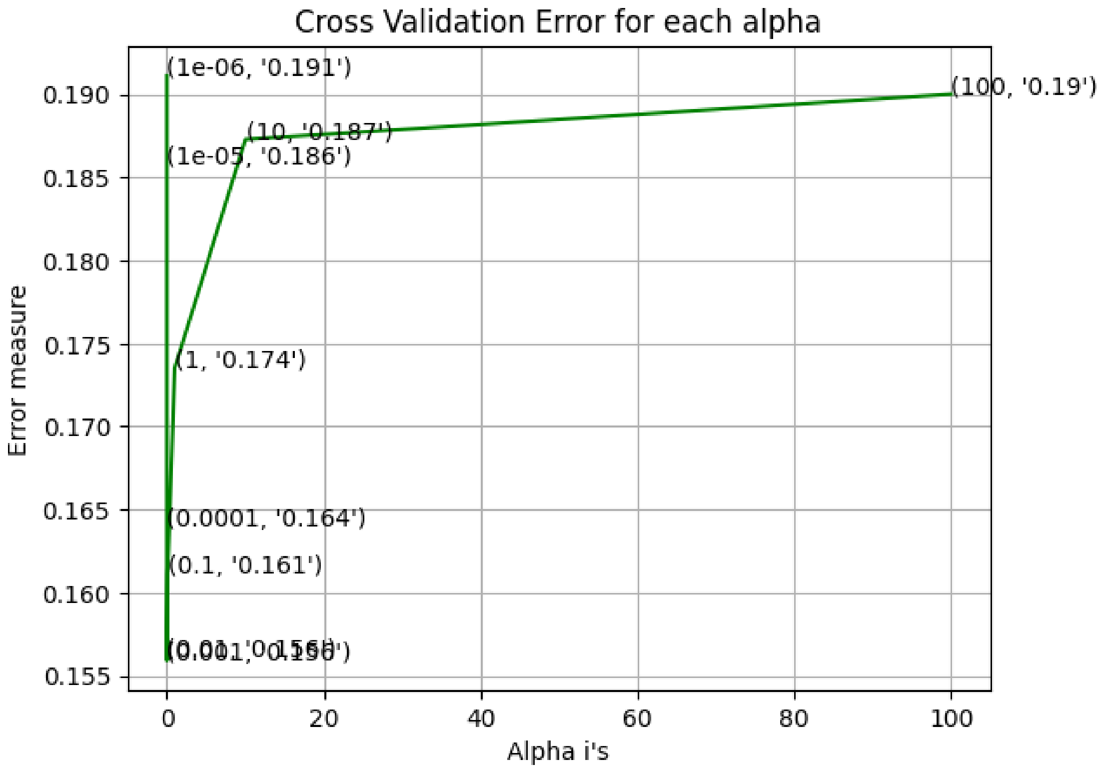
With Class balancing

Hyper paramter tuning

```
[ ]: #Logistic Regression
#With Class balancing
#Hyper paramter tuning



alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',␣
 ↪loss='log', random_state=42)
    clf.fit(train_df,y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df, y_train)
    sig_clf_probs = sig_clf.predict_proba(cv_df)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use␣
 ↪log-probability estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
clf.fit(train_df, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df, y_train)

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

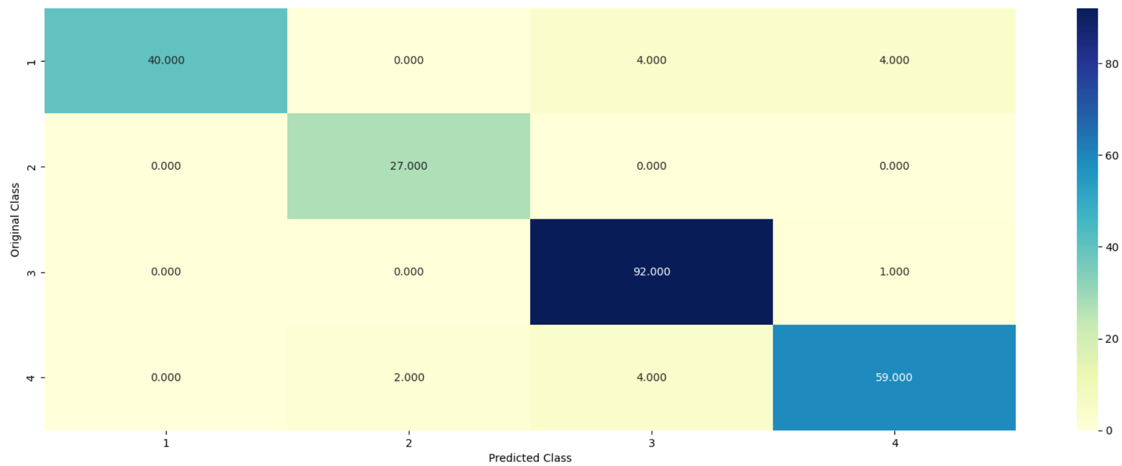
```
for alpha = 1e-06
Log Loss : 0.19113171780419833
for alpha = 1e-05
Log Loss : 0.18584508186731702
for alpha = 0.0001
Log Loss : 0.16403903917983137
for alpha = 0.001
Log Loss : 0.1559052882112476
for alpha = 0.01
Log Loss : 0.15610417279373368
for alpha = 0.1
Log Loss : 0.161110735323708
for alpha = 1
Log Loss : 0.17351882883436703
for alpha = 10
Log Loss : 0.18729152018312847
for alpha = 100
Log Loss : 0.19001110697968235
```



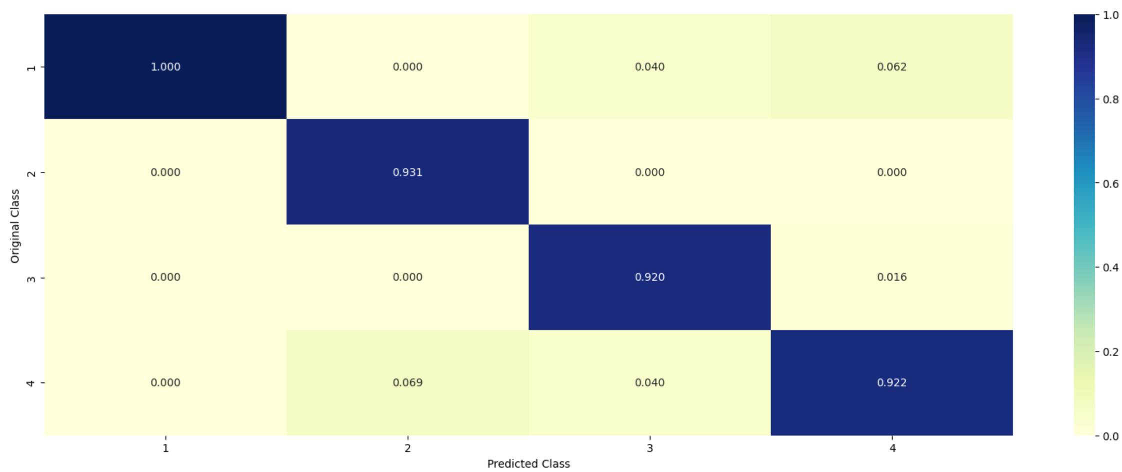

```
For values of best alpha =  0.001 The train log loss is: 0.1727204896809822
For values of best alpha =  0.001 The cross validation log loss is:
0.1559052882112476
For values of best alpha =  0.001 The test log loss is: 0.1661671384860418
```

[ ]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
↪penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_df.values, y_train.values, cv_df.
↪values, y_cv.values, clf)
```

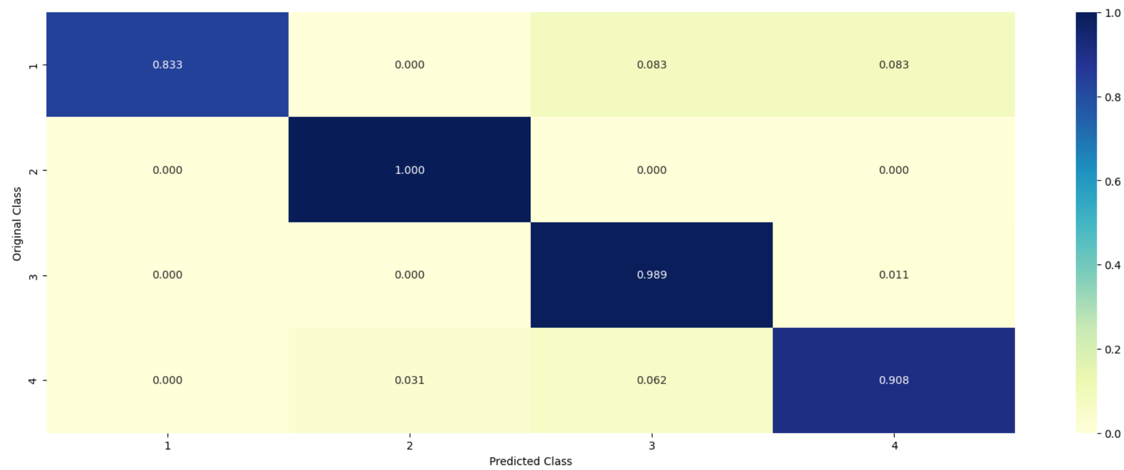
```
Log loss : 0.1559052882112476
Number of mis-classified points : 163.63090128755366
-------------------- Confusion matrix --------------------
```



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



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



Random Forest Classifier

Hyper paramter tuning

[ ]:
```
import pickle
pickle.dump(sig_clf, open('/content/drive/MyDrive/final_prediction.pickle',␣
↪'wb'))
pickle.dump(scaler, open('/content/drive/MyDrive/scaler.pickle', 'wb'))


alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',␣
↪max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_df,y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_df,y_train)
        sig_clf_probs = sig_clf.predict_proba(cv_df)
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.
↪classes_, eps=1e-15))
        print("Log Loss :",log_loss(y_cv, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
```

```
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),␣
 ↪(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
 ↪n_jobs=-1)
clf.fit(train_df,y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df,y_train)

predict_y = sig_clf.predict_proba(train_df)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train␣
 ↪log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_df)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross␣
 ↪validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
predict_y = sig_clf.predict_proba(test_df)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test␣
 ↪log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

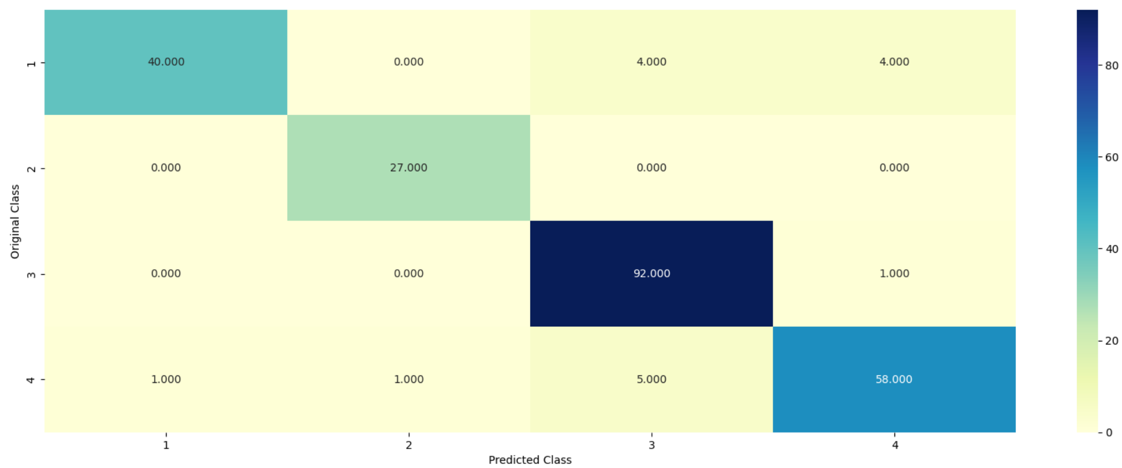
```
for n_estimators = 100 and max depth =  5
Log Loss : 0.17485643131115827
for n_estimators = 100 and max depth =  10
Log Loss : 0.17689845098260562
for n_estimators = 200 and max depth =  5
Log Loss : 0.1749253413215618
for n_estimators = 200 and max depth =  10
Log Loss : 0.17412827734558625
for n_estimators = 500 and max depth =  5
Log Loss : 0.1679919273674284
for n_estimators = 500 and max depth =  10
Log Loss : 0.17209225434292702
for n_estimators = 1000 and max depth =  5
Log Loss : 0.16512473923084903
for n_estimators = 1000 and max depth =  10
Log Loss : 0.17073808391824918
for n_estimators = 2000 and max depth =  5
Log Loss : 0.16457364744805636
for n_estimators = 2000 and max depth =  10
```

```
Log Loss : 0.17070997401001095
For values of best estimator =  2000 The train log loss is: 0.15018613000009057
For values of best estimator =  2000 The cross validation log loss is:
0.164573647448056 36
For values of best estimator =  2000 The test log loss is: 0.14752391650359561
```

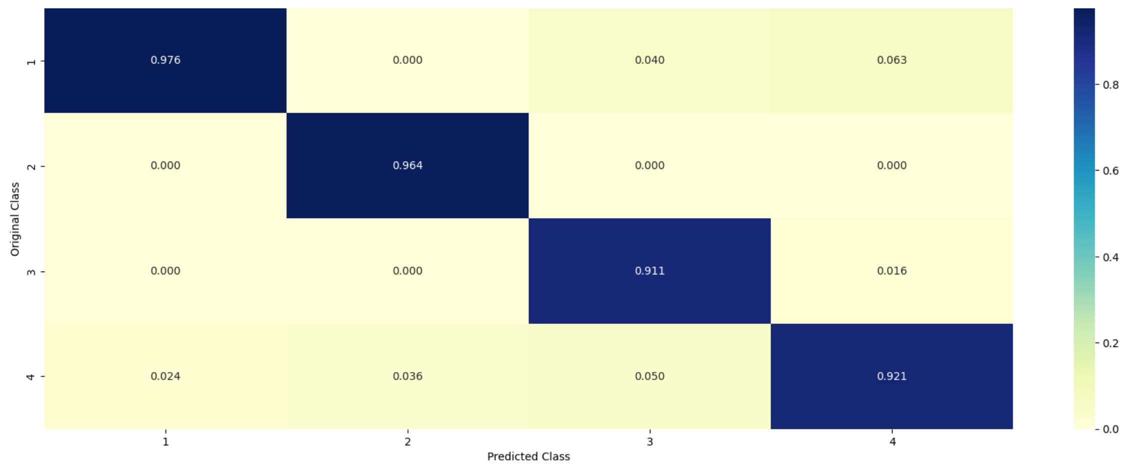
Testing model with best hyper parameters

```
[ ]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
      ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
      ↪n_jobs=-1)
     # predict_and_plot_confusion_matrix(train_x_onehotCoding,␣
      ↪train_y,cv_x_onehotCoding,cv_y, clf)
     predict_and_plot_confusion_matrix(train_df.values, y_train.values, cv_df.
      ↪values, y_cv.values, clf)
```

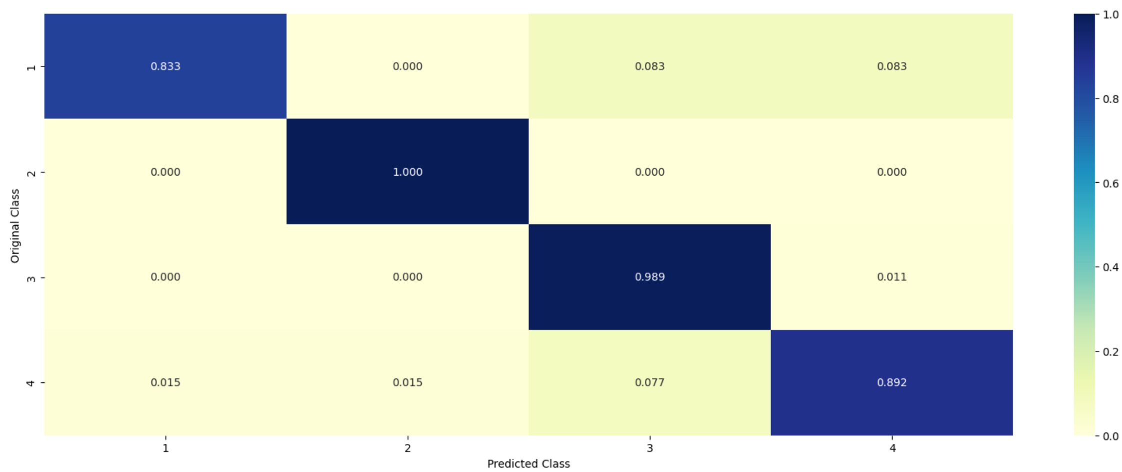
```
Log loss : 0.16457364744805636
Number of mis-classified points : 163.42060085836908
-------------------- Confusion matrix --------------------
```




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




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



##Conclusions: 1. Among 3 ML Models , Random Forest is the best ML Model for our task. 2. Train and test performance of Model can be furthur improved by using Deep Learning models ,However at the Cost of computational expense.

[ ]:
```
import pickle
pickle.dump(sig_clf, open('/content/drive/MyDrive/final_prediction.pickle',
    'wb'))
pickle.dump(scaler, open('/content/drive/MyDrive/scaler.pickle', 'wb'))
```

[ ]:

[ ]: