3.6 Featurizing text data with tfidf weighted word-vectors

In [1]:
```
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.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy
```

In [2]:
```
# avoid decoding problems
df = pd.read_csv("train.csv")

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf
-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf
-8"))
```

```
# ----------------- python 3 ---------------------
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [3]:
```
df.head()
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [4]:
```
df['question'] = df['question1'] + df['question2']
from sklearn.model_selection import train_test_split
y_true = df['is_duplicate']

train_df,test_df, y_train, y_test = train_test_split(df, y_true, strati
fy=y_true, test_size=0.3)
text_vec = TfidfVectorizer(min_df=3)
train_df['question'] = train_df['question1'] + train_df['question2']
train_question_feature_onehotCoding = text_vec.fit_transform(train_df[
'question'])


# train_question_feature_onehotCoding.sum(axis=0).A1 will sum every row
 and returns (1*number of features) vector
train_question_fea_counts = train_question_feature_onehotCoding.sum(axi
```

```
s=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(text_vec.get_feature_names()),train_quest
ion_fea_counts))


print("Total number of unique words in train data :", len(text_vec.get_
feature_names()))
```

```
Total number of unique words in train data : 31673
```

In [5]:
```
# don't forget to normalize every feature
train_question_feature_onehotCoding = normalize(train_question_feature_
onehotCoding, axis=0)
train_question_feature_onehotCoding.shape
# we use the same vectorizer that was trained on train data
test_df['question'] = test_df['question1'] + test_df['question2']
test_question_feature_onehotCoding = text_vec.transform(test_df['questi
on'])
# don't forget to normalize every feature
test_question_feature_onehotCoding = normalize(test_question_feature_on
ehotCoding, axis=0)
```

In [6]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
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 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



from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

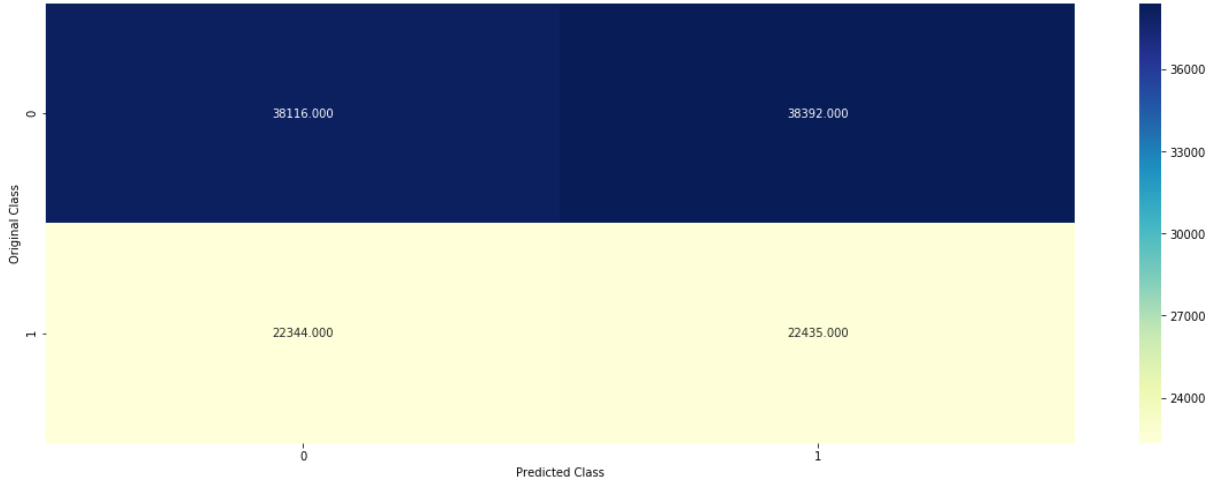
- After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores.
- here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usage/vectors-similarity
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [7]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)


    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))


    labels = [0,1]
    # 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=la
bels, 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=la
bels, 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=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

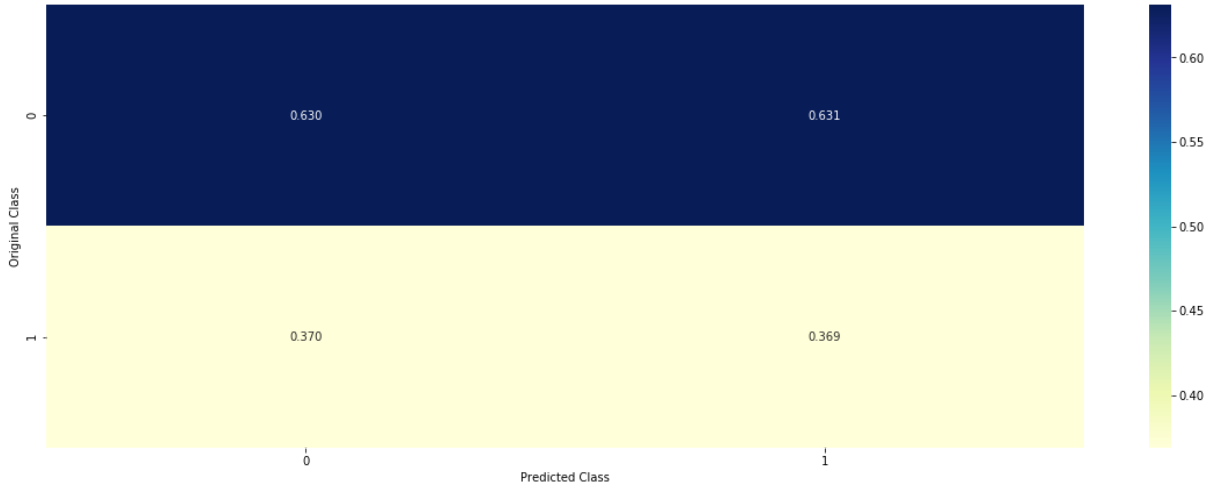
Random Model

In [9]:
```
test_data_len = test_df.shape[0]

# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,2))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,2)
    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_p
redicted_y, eps=1e-15))

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

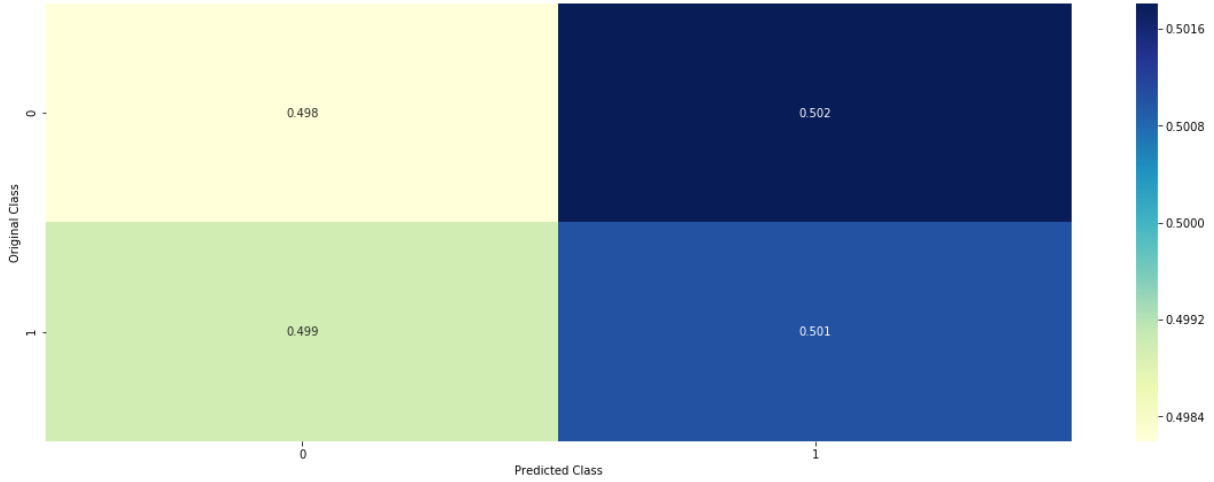
```
Log loss on Test Data using Random Model 0.8860818681047058
-------------------- Confusion matrix --------------------
```




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



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



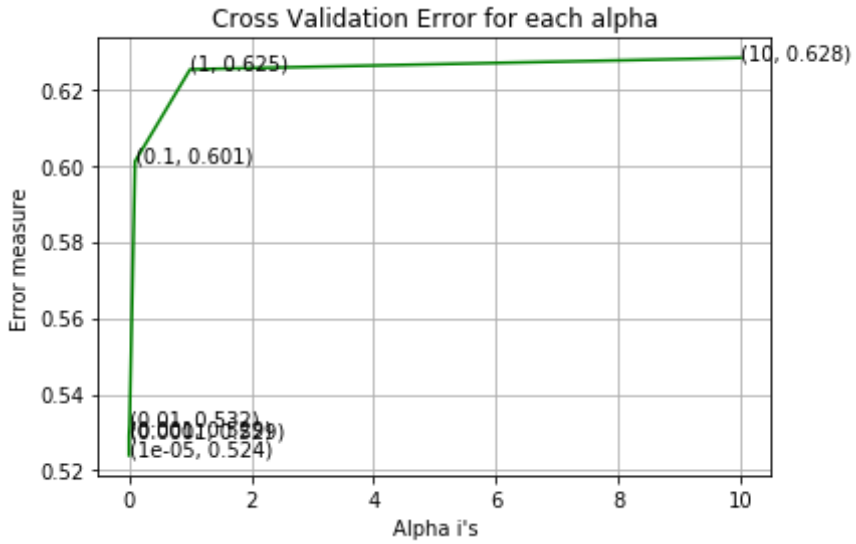
Logistic Regression with hyperparameter tuning

In [14]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
```

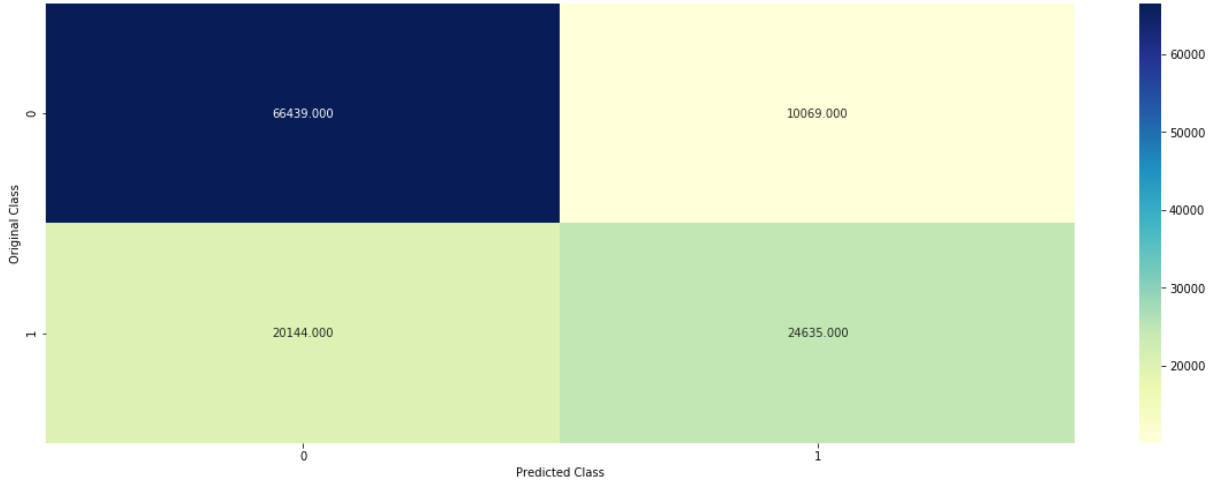
```
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_question_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_question_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(test_question_feature_onehotCodin
g)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, predict_y, labels=clf.classes_, eps=1e-15))

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

```
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],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(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_question_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_question_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_question_feature_onehotCoding)
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(test_question_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

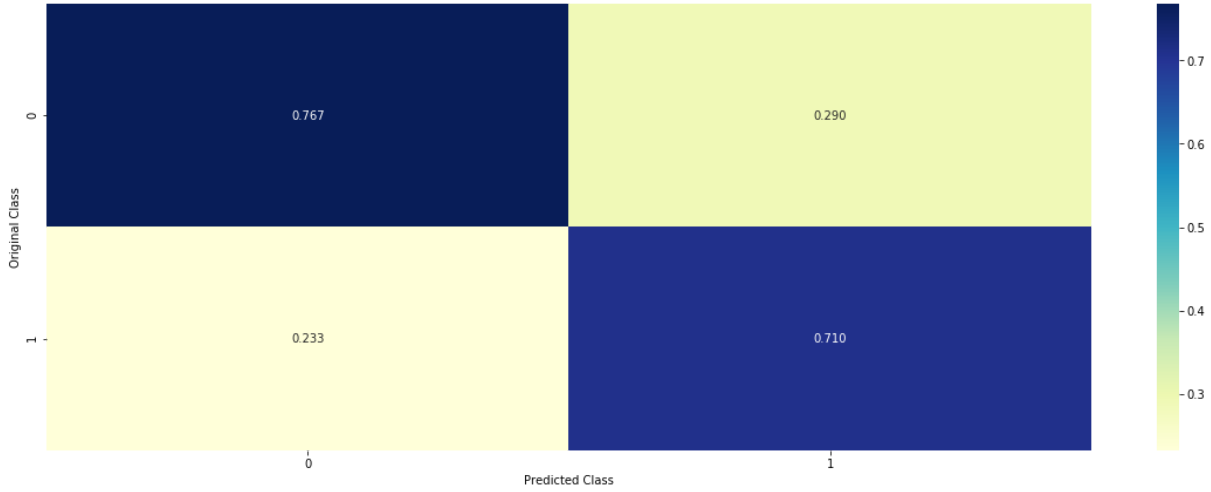
```
For values of alpha =  1e-05 The log loss is: 0.5239947922899115
For values of alpha =  0.0001 The log loss is: 0.5288303511619382
For values of alpha =  0.001 The log loss is: 0.5294648861573048
For values of alpha =  0.01 The log loss is: 0.531914519307494
For values of alpha =  0.1 The log loss is: 0.6010244666144026
For values of alpha =  1 The log loss is: 0.6251590753874883
For values of alpha =  10 The log loss is: 0.6281789132273929
```



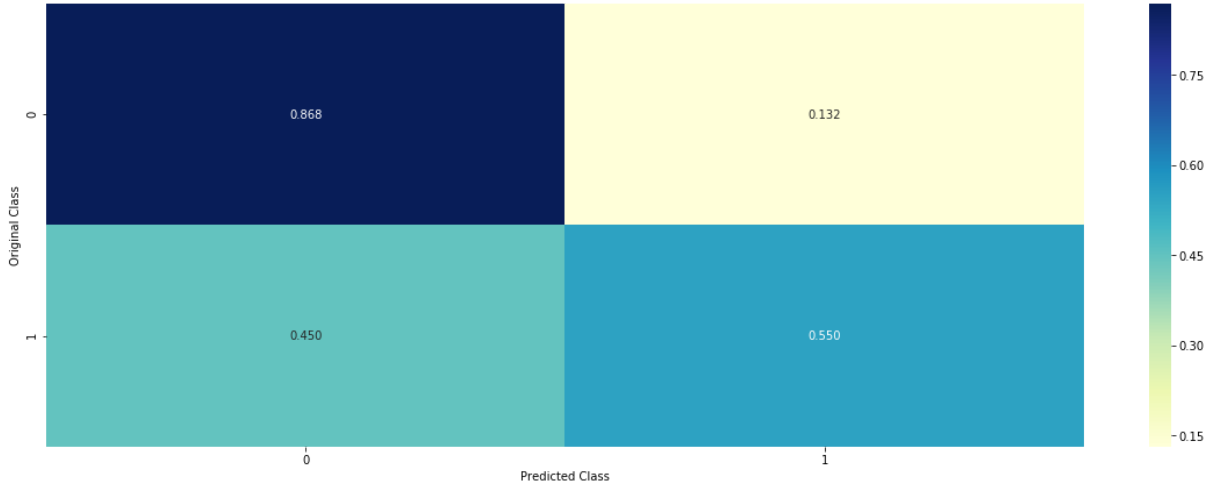

```
For values of best alpha =  1e-05 The train log loss is: 0.470735443278
9601
For values of best alpha =  1e-05 The test log loss is: 0.5239947922899
115
Total number of data points : 121287
-------------------- Confusion matrix --------------------
```



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




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

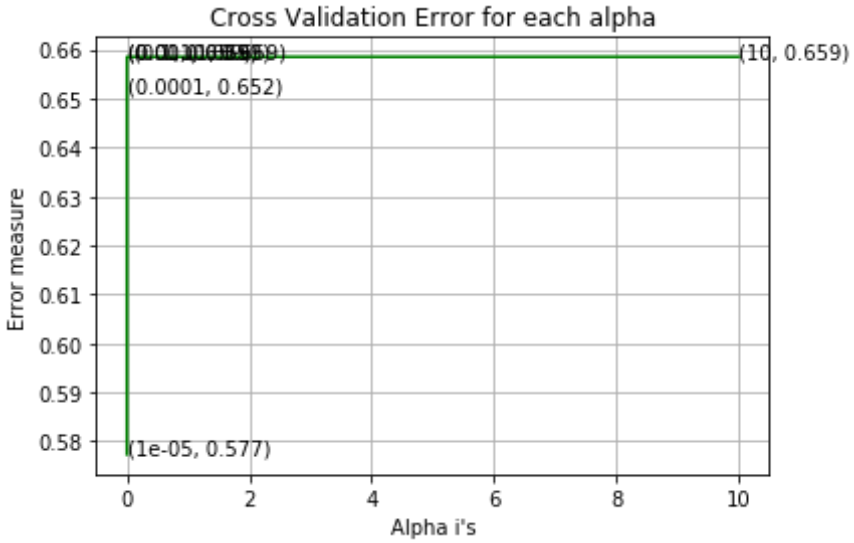


Linear SVM with hyperparameter tuning

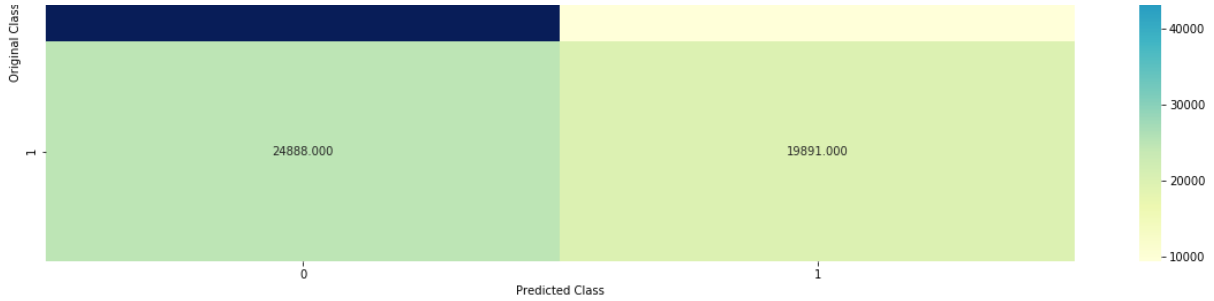
In [15]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_sta
te=42)
    clf.fit(train_question_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_question_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(test_question_feature_onehotCodin
g)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
```

```
st, predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],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(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l1', loss='hinge'
, random_state=42)
clf.fit(train_question_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_question_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_question_feature_onehotCoding)
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(test_question_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

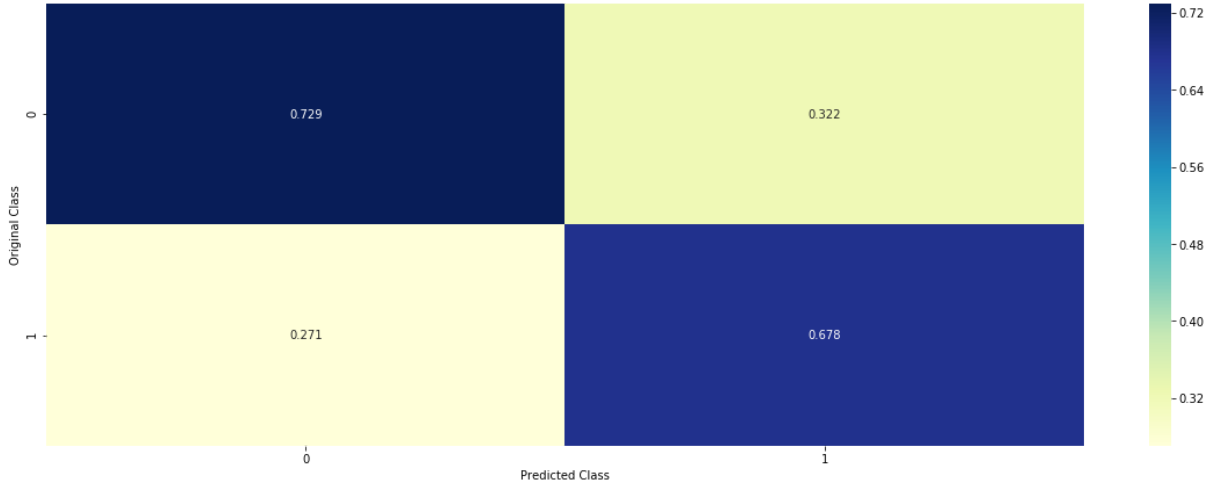
```
For values of alpha =  1e-05 The log loss is: 0.5773318558627232
For values of alpha =  0.0001 The log loss is: 0.6515494289228502
For values of alpha =  0.001 The log loss is: 0.6585278256322724
For values of alpha =  0.01 The log loss is: 0.6585278256322724
For values of alpha =  0.1 The log loss is: 0.6585278256322723
For values of alpha =  1 The log loss is: 0.658527825632263
For values of alpha =  10 The log loss is: 0.6585278256322629
```




```
For values of best alpha =  1e-05 The train log loss is: 0.558576034361
1987
For values of best alpha =  1e-05 The test log loss is: 0.5773318558627
232
Total number of data points : 121287
-------------------- Confusion matrix --------------------
```



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



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

XGBoost

In [26]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
xgb_model = xgb.XGBClassifier()

parameters = {
              'objective':['binary:logistic'],
              'gamma': [0.5, 2 , 2.5 , 4, 5],
              'max_depth': [3, 4, 5,6],
              'n_estimators': [5,6,7],
              }


clf = RandomizedSearchCV(xgb_model, parameters, cv=5,scoring="neg_log_loss")
clf.fit(X=train_question_feature_onehotCoding, y=y_train)
print(clf.best_estimator_)
print (clf.best_score_, clf.best_params_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=2.5, learning_rate=0.1, max_delta_step=0,
       max_depth=6, min_child_weight=1, missing=None, n_estimators=7,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
-0.6302398891112505 {'objective': 'binary:logistic', 'n_estimators': 7, 'max_depth': 6, 'gamma': 2.5}
```

In [28]:
```
import xgboost as xgb
```

```
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 6
params['n_estimators'] = 7
params['gamma'] = 2.5
params['silent']= 1
d_train = xgb.DMatrix(train_question_feature_onehotCoding, label=y_trai
n)
d_test = xgb.DMatrix(test_question_feature_onehotCoding, label=y_test)

watchlist = [(d_train, 'train'), (d_test, 'valid')]

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=
20, verbose_eval=10)

xgdmat = xgb.DMatrix(train_question_feature_onehotCoding,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))
```

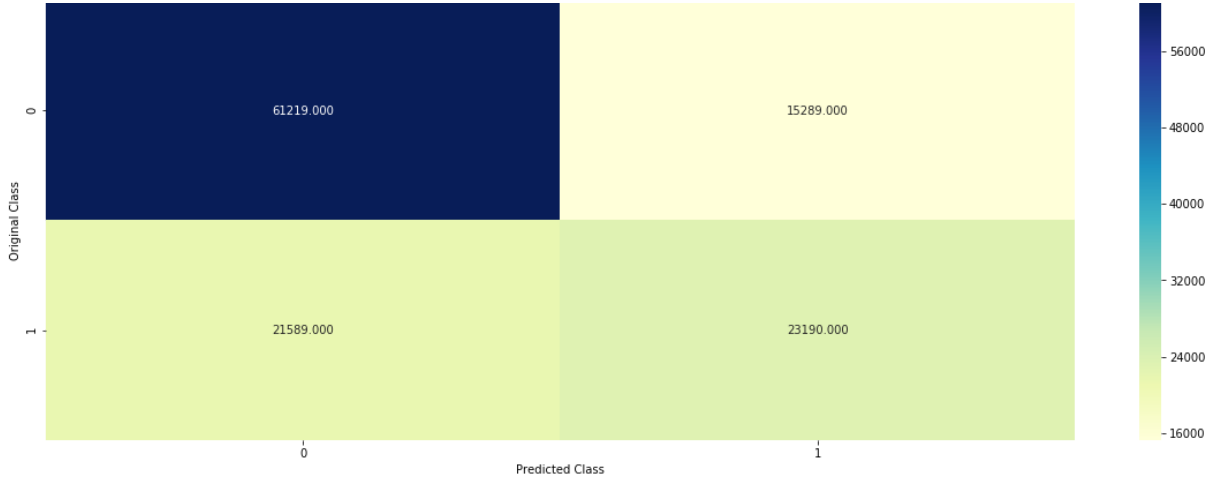
```
[0]     train-logloss:0.690184  valid-logloss:0.691146
Multiple eval metrics have been passed: 'valid-logloss' will be used fo
r early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.666003  valid-logloss:0.675476
[20]    train-logloss:0.648277  valid-logloss:0.663629
[30]    train-logloss:0.634516  valid-logloss:0.653721
[40]    train-logloss:0.623921  valid-logloss:0.646545
[50]    train-logloss:0.615155  valid-logloss:0.64003
[60]    train-logloss:0.608112  valid-logloss:0.6346
[70]    train-logloss:0.601646  valid-logloss:0.629661
[80]    train-logloss:0.596293  valid-logloss:0.625466
[90]    train-logloss:0.591683  valid-logloss:0.622012
[100]   train-logloss:0.587388  valid-logloss:0.618689
[110]   train-logloss:0.583647  valid-logloss:0.615773
[120]   train-logloss:0.58009   valid-logloss:0.613227
[130]   train-logloss:0.576981  valid-logloss:0.610828
```

```
[140]   train-logloss:0.574236  valid-logloss:0.608788
[150]   train-logloss:0.571578  valid-logloss:0.606676
[160]   train-logloss:0.568934  valid-logloss:0.604979
[170]   train-logloss:0.566456  valid-logloss:0.603397
[180]   train-logloss:0.56421   valid-logloss:0.601755
[190]   train-logloss:0.562146  valid-logloss:0.600319
[200]   train-logloss:0.560112  valid-logloss:0.598948
[210]   train-logloss:0.558358  valid-logloss:0.597808
[220]   train-logloss:0.556468  valid-logloss:0.596705
[230]   train-logloss:0.554825  valid-logloss:0.5956
[240]   train-logloss:0.553162  valid-logloss:0.59468
[250]   train-logloss:0.551531  valid-logloss:0.593654
[260]   train-logloss:0.550077  valid-logloss:0.592759
[270]   train-logloss:0.548628  valid-logloss:0.592016
[280]   train-logloss:0.547172  valid-logloss:0.59133
[290]   train-logloss:0.545889  valid-logloss:0.590634
[300]   train-logloss:0.544669  valid-logloss:0.589897
[310]   train-logloss:0.543475  valid-logloss:0.589301
[320]   train-logloss:0.54232   valid-logloss:0.588601
[330]   train-logloss:0.541228  valid-logloss:0.588044
[340]   train-logloss:0.5402    valid-logloss:0.587419
[350]   train-logloss:0.539187  valid-logloss:0.586871
[360]   train-logloss:0.538164  valid-logloss:0.586353
[370]   train-logloss:0.537182  valid-logloss:0.585944
[380]   train-logloss:0.536319  valid-logloss:0.585551
[390]   train-logloss:0.535377  valid-logloss:0.584989
[399]   train-logloss:0.534606  valid-logloss:0.584545
The test log loss is: 0.5845454278589597
```

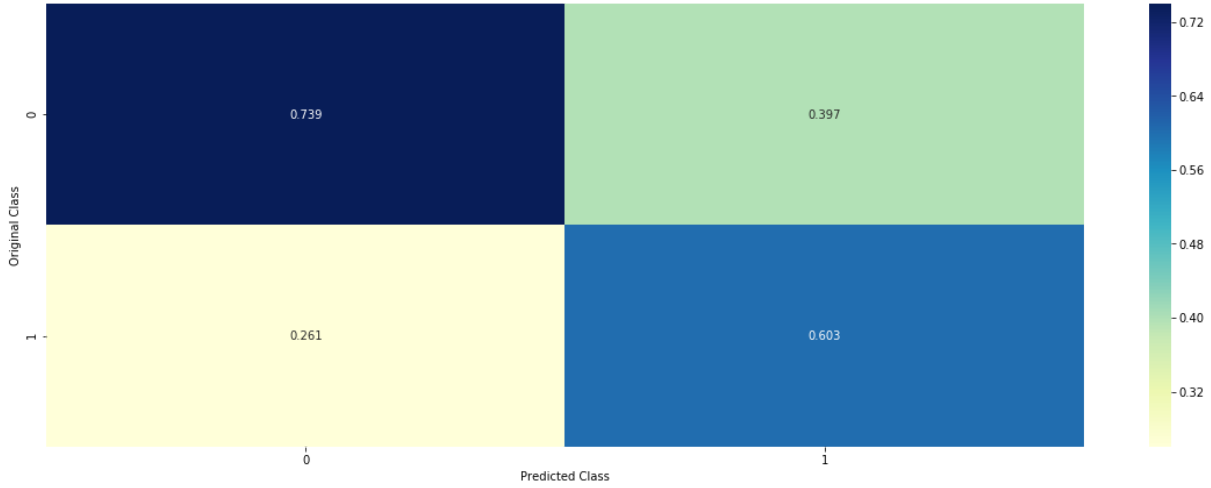
In [29]:

```
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

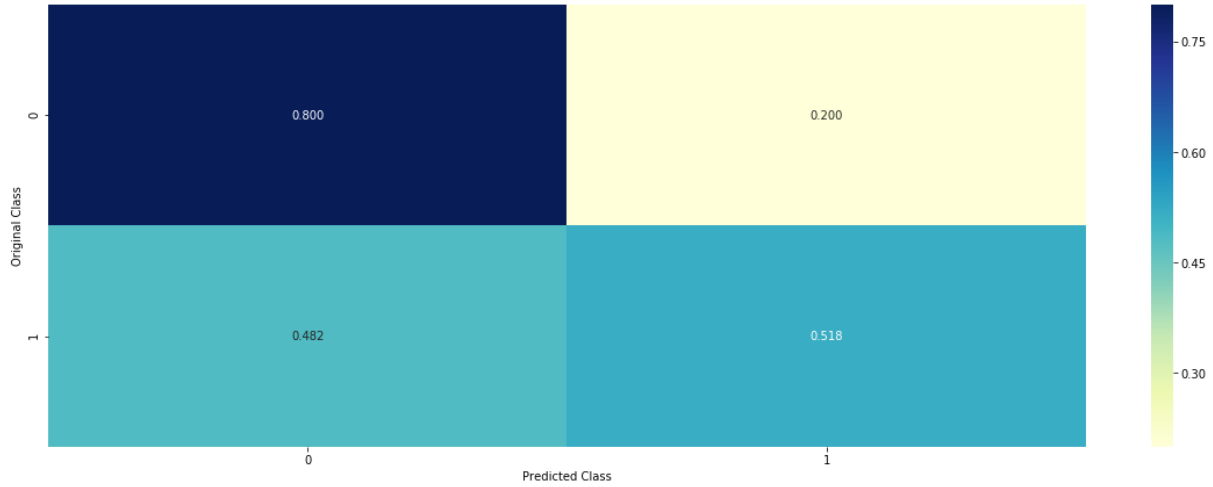
```
Total number of data points : 121287
-------------------- Confusion matrix --------------------
```



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



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



In [1]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","Train  loss","Test loss"]

x.add_row(["Logistic Regression",".47",".523"])
x.add_row(["Linear SVM",".55",".577"])
x.add_row(["XGBOOST",".53",".584"])


print(x)
```

```
+---------------------+-------------+-----------+
|        Model        | Train  loss | Test loss |
+---------------------+-------------+-----------+
| Logistic Regression |     .47     |    .523   |
|      Linear SVM     |     .55     |    .577   |


|       XGBOOST       |     .53     |    .584   |
```

+----------------------+-------------+-----------+

Conclusion

1)Exploratory Data analysis on the model 2)Adding the basic features like frequency of words , common words etc 3)Adding advanced features and applying EDA on them 4)Applying TFIDF on the q1 + q2 and giving them as features after splitting in to train and test 5)Doing hyperparameter tuning and applying those models.. 6)Comparing all the models