
# Quora Question Pairs

## 1. Business Problem

### 1.1 Description

Quora is a place to gain and share knowledge—about anything. It’s a platform to ask questions and connect with people who contribute unique insights and quality answers. This empowers people to learn from each other and to better understand the world.

Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

> Credits: Kaggle

__ Problem Statement __

- Identify which questions asked on Quora are duplicates of questions that have already been asked.
- This could be useful to instantly provide answers to questions that have already been answered.
- We are tasked with predicting whether a pair of questions are duplicates or not.

## 1.2 Sources/Useful Links

- Source : https://www.kaggle.com/c/quora-question-pairs (https://www.kaggle.com/c/quora-question-pairs)

**__ Useful Links __**

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments (https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments)
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0 (https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0)
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning (https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning)
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30 (https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30)

## 1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. No strict latency concerns.
4. Interpretability is partially important.

# 2. Machine Learning Probelm

## 2.1 Data

### 2.1.1 Data Overview

- Data will be in a file Train.csv
- Train.csv contains 5 columns : qid1, qid2, question1, question2, is_duplicate
- Size of Train.csv - 60MB
- Number of rows in Train.csv = 404,290

### 2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in
india?","What is the step by step guide to invest in share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamond?","What w
ould happen if the Indian government stole the Kohinoor (Koh-i-Noor) diam
ond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a
great geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I s
ee all my Youtube comments?","1"
```

## 2.2 Mapping the real world problem to an ML problem

### 2.2.1 Type of Machine Leaning Problem

It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.

### 2.2.2 Performance Metric

Source: https://www.kaggle.com/c/quora-question-pairs#evaluation (https://www.kaggle.com/c/quora-question-pairs#evaluation)

Metric(s):

- log-loss : https://www.kaggle.com/wiki/LogarithmicLoss (https://www.kaggle.com/wiki/LogarithmicLoss)
- Binary Confusion Matrix

## 2.3 Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

# 3. Exploratory Data Analysis

In [6]:

```
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc
import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
import re
# This package is used for finding longest common subsequence between two stri
# you can write your own dp code for this
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-pyt
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
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.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
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
```

## 3.1 Reading data and basic stats

In [7]:
```
df = pd.read_csv("train.csv")

print("Number of data points:",df.shape[0])
```

```
Number of data points: 404290
```

In [8]:
```
df.head()
```

Out[8]:

| | 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 [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id              404290 non-null int64
qid1            404290 non-null int64
qid2            404290 non-null int64
question1       404290 non-null object
question2       404288 non-null object
is_duplicate    404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

We are given a minimal number of data fields here, consisting of:

- id: Looks like a simple rowID

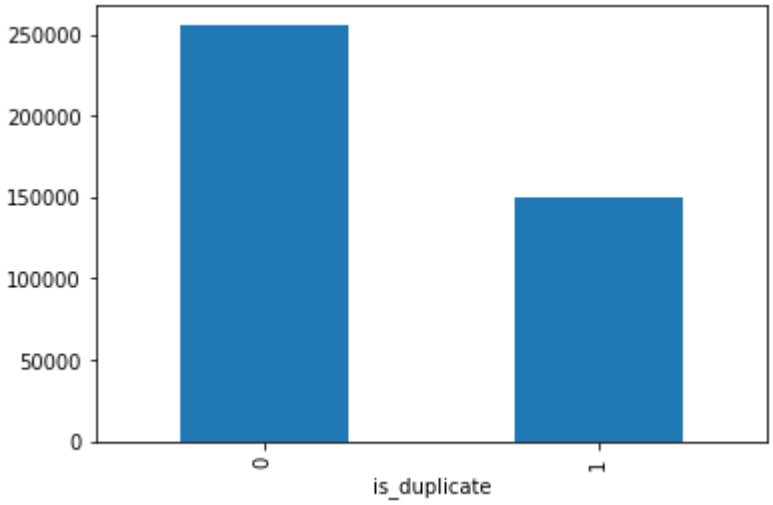
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

### 3.2.1 Distribution of data points among output classes

- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [10]:
```
1 df.groupby("is_duplicate")['id'].count().plot.bar()
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22b00727d30>
```




In [11]:
```
1 print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

```
~> Total number of question pairs for training:
   404290
```

In [12]:
```
1 print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(1
2 print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(rou
```

```
~> Question pairs are not Similar (is_duplicate = 0):
   63.08%

~> Question pairs are Similar (is_duplicate = 1):
   36.92%
```

### 3.2.2 Number of unique questions

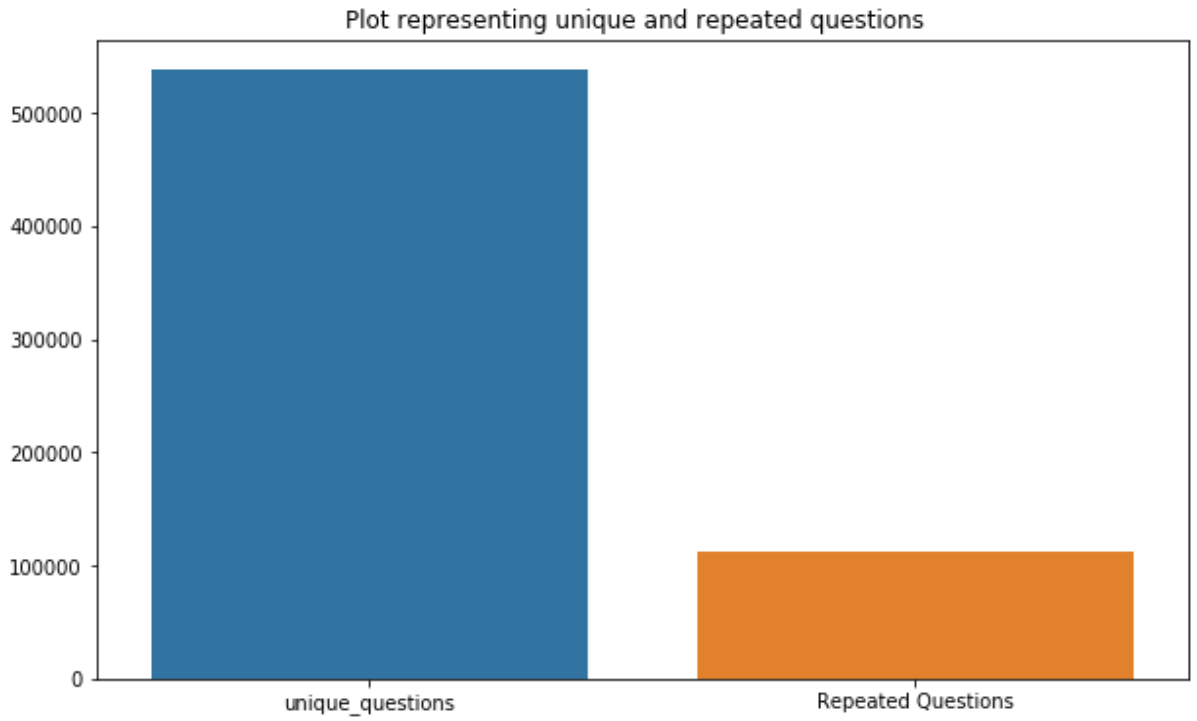
In [13]:

```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

print ('Number of unique questions that appear more than one time: {} ({}%)\n'

print ('Max number of times a single question is repeated: {}\n'.format(max(qi

q_vals=qids.value_counts()

q_vals=q_vals.values
```

```
Total num of  Unique Questions are: 537933

Number of unique questions that appear more than one time: 111780 (20.77953945
937505%)

Max number of times a single question is repeated: 157
```

In [14]:

```

x = ["unique_questions" , "Repeated Questions"]
y =  [unique_qs , qs_morethan_onetime]

plt.figure(figsize=(10, 6))
plt.title ("Plot representing unique and repeated questions  ")
sns.barplot(x,y)
plt.show()
```



### 3.2.3 Checking for Duplicates

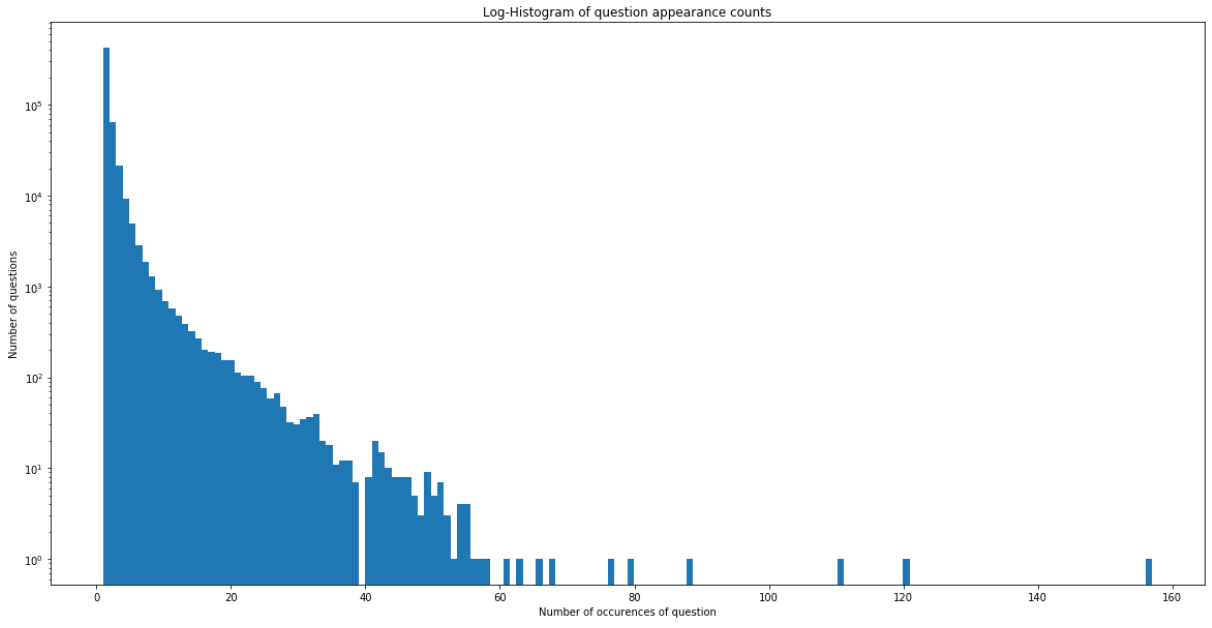
In [15]:
```
#checking whether there are any repeated pair of questions

pair_duplicates = df[['qid1','qid2','is_duplicate']].groupby(['qid1','qid2']).

print ("Number of duplicate questions",(pair_duplicates).shape[0] - df.shape[0
```

```
Number of duplicate questions 0
```

### 3.2.4 Number of occurrences of each question

In [24]:
```
plt.figure(figsize=(20, 10))

plt.hist(qids.value_counts(), bins=160)

plt.yscale('log', nonposy='clip')

plt.title('Log-Histogram of question appearance counts')

plt.xlabel('Number of occurences of question')

plt.ylabel('Number of questions')

print ('Maximum number of times a single question is repeated: {}\n'.format(ma
```

```
Maximum number of times a single question is repeated: 157
```




### 3.2.5 Checking for NULL values

In [17]:
```
#Checking whether there are any rows with null values
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

```
            id    qid1    qid2                         question1 question2  \
105780  105780  174363  174364    How can I develop android app?       NaN
201841  201841  303951  174364  How can I create an Android app?       NaN

        is_duplicate
105780             0
201841             0
```

- There are two rows with null values in question2

In [18]:
```
# Filling the null values with ' '
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
```

## 3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1
- **q2len** = Length of q2
- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2
- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)
- **word_share** = (word_common)/(word_Total)
- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2
- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

In [20]:

```
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1
else:
    df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
    df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
    df['q1len'] = df['question1'].str.len()
    df['q2len'] = df['question2'].str.len()
    df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

    def normalized_word_Common(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split
        return 1.0 * len(w1 & w2)
    df['word_Common'] = df.apply(normalized_word_Common, axis=1)

    def normalized_word_Total(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split
        return 1.0 * (len(w1) + len(w2))
    df['word_Total'] = df.apply(normalized_word_Total, axis=1)

    def normalized_word_share(row):
        w1 = set(map(lambda word: word.lower().strip(), row['question1'].split
        w2 = set(map(lambda word: word.lower().strip(), row['question2'].split
        return 1.0 * len(w1 & w2)/(len(w1) + len(w2))
    df['word_share'] = df.apply(normalized_word_share, axis=1)

    df['freq_q1+q2'] = df['freq_qid1']+df['freq_qid2']
    df['freq_q1-q2'] = abs(df['freq_qid1']-df['freq_qid2'])

    df.to_csv("df_fe_without_preprocessing_train.csv", index=False)

df.head()
```

Out[20]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_v |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 66 | 57 | |
| **1** | 1 | 3 | 4 | What is the story of Kohinoor (Koh-i-Noor) Dia... | What would happen if the Indian government sto... | 0 | 4 | 1 | 51 | 88 | |
| **2** | 2 | 5 | 6 | How can I increase the speed of my internet co... | How can Internet speed be increased by hacking... | 0 | 1 | 1 | 73 | 59 | |

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_v |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **3** | 3 | 7 | 8 | Why am I mentally very lonely? How can I solve... | Find the remainder when [math]23^{24} [/math] i... | 0 | 1 | 1 | 50 | 65 | |
| **4** | 4 | 9 | 10 | Which one dissolve in water quikly sugar, salt... | Which fish would survive in salt water? | 0 | 3 | 1 | 76 | 39 | |

### 3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

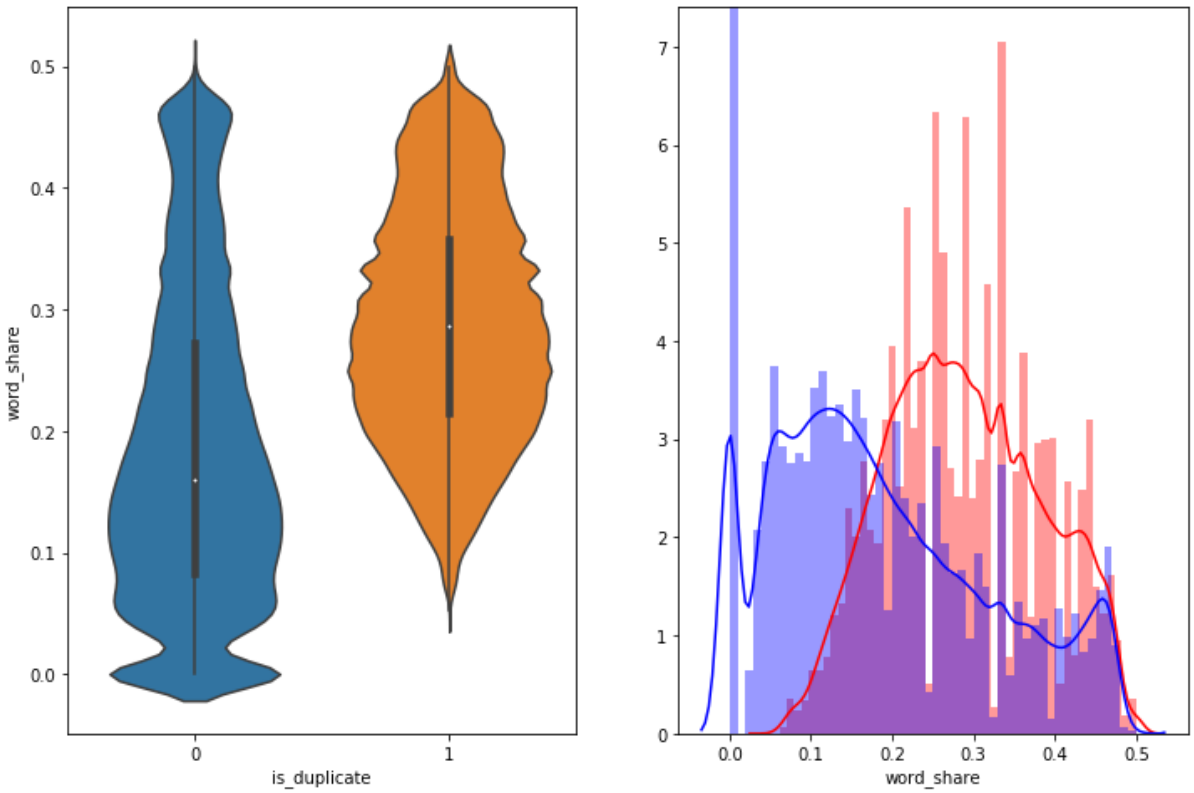
In [21]:
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words'

print ("Minimum length of the questions in question2 : " , min(df['q2_n_words'

print ("Number of Questions with minimum length [question1] :", df[df['q1_n_wo
print ("Number of Questions with minimum length [question2] :", df[df['q2_n_wo
```

```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
Number of Questions with minimum length [question1] : 67
Number of Questions with minimum length [question2] : 24
```

#### 3.3.1.1 Feature: word_share

In [25]:

```
plt.figure(figsize=(12, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_share', data = df[0:])

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_share'][0:] , label = "1", co
sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , c
plt.show()
```

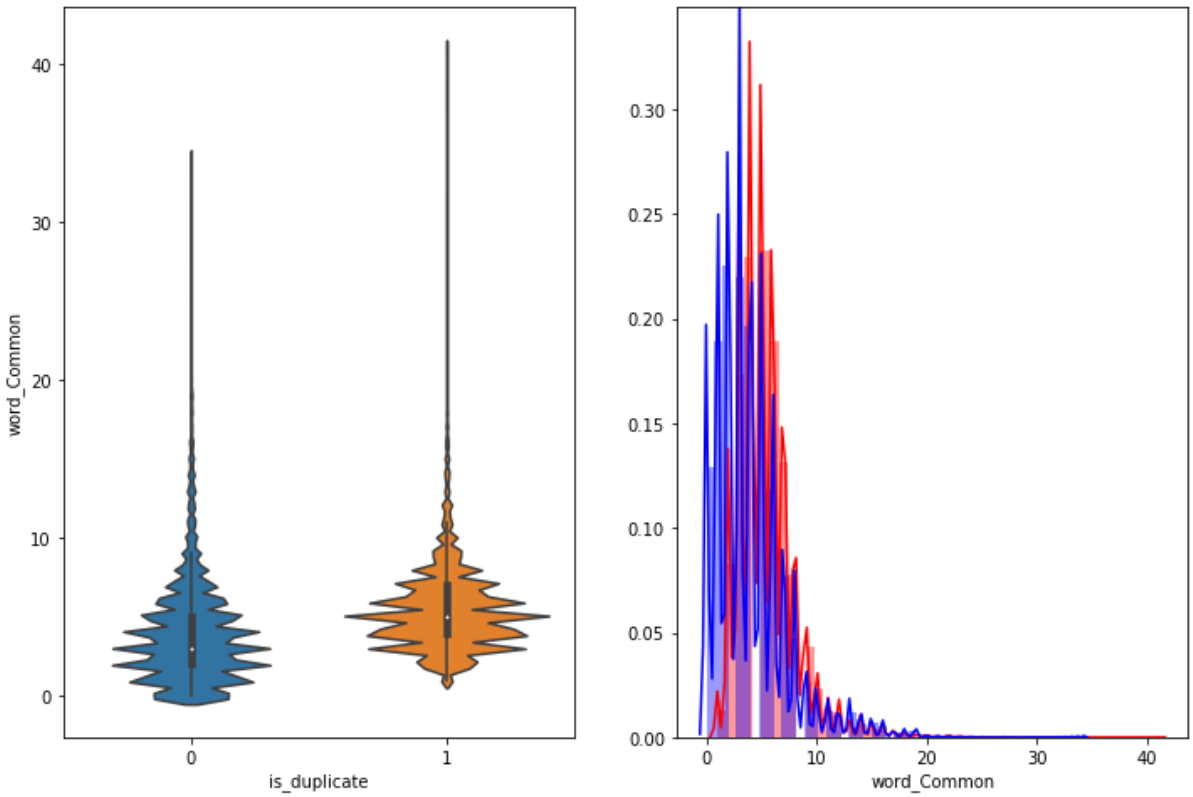



- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

#### 3.3.1.2 Feature: word_Common

In [26]:

```
plt.figure(figsize=(12, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'word_Common', data = df[0:])

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['word_Common'][0:] , label = "1", c
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label = "0" ,
plt.show()
```




The distributions of the word_Common feature in similar and non-similar questions are highly overlapping

## Data Pre processing

In [7]:

```
#https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-co
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the pre
```

In [8]:
```
1 df.head(2)
```

Out[8]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_wo |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 66 | 57 | |
| **1** | 1 | 3 | 4 | What is the story of Kohinoor (Koh-i-Noor) Dia... | What would happen if the Indian government sto... | 0 | 4 | 1 | 51 | 88 | |

## 3.4 Preprocessing of Text

- Preprocessing:
  - Removing html tags
  - Removing Punctuations
  - Performing stemming
  - Removing Stopwords
  - Expanding contractions etc.

In [9]:

```
# To get the results in 4 decemal points
SAFE_DIV = 0.0001

STOP_WORDS = stopwords.words("english")


def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000", "m").replace(",000", "k").replace("’", "'").repl
                           .replace("won't", "will not").replace("cannot", "ca
                           .replace("n't", " not").replace("what's", "what is'
                           .replace("'ve", " have").replace("i'm", "i am").rep
                           .replace("he's", "he is").replace("she's", "she is'
                           .replace("%", " percent ").replace("₹", " rupee ").
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


    porter = PorterStemmer()
    pattern = re.compile('\W')

    if type(x) == type(''):
        x = re.sub(pattern, ' ', x)


    if type(x) == type(''):
        x = porter.stem(x)
        example1 = BeautifulSoup(x)
        x = example1.get_text()


    return x

```

- Function to Compute and get the features : With 2 parameters of Question 1 and Question 2

## 3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

- **Token**: You get a token by splitting sentence a space
- **Stop_Word** : stop words as per NLTK.
- **Word** : A token that is not a stop_word

Features:

- **cwc_min** : Ratio of common_word_count to min lenghth of word count of Q1 and Q2
  cwc_min = common_word_count / (min(len(q1_words), len(q2_words))

- **cwc_max** : Ratio of common_word_count to max lenghth of word count of Q1 and Q2
  cwc_max = common_word_count / (max(len(q1_words), len(q2_words))

- **csc_min** : Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2
  csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))

- **csc_max** : Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2
  csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))

- **ctc_min** : Ratio of common_token_count to min lenghth of token count of Q1 and Q2
  ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))

- **ctc_max** : Ratio of common_token_count to max lenghth of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

- **last_word_eq** : Check if First word of both questions is equal or not
  last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])

- **first_word_eq** : Check if First word of both questions is equal or not
  first_word_eq = int(q1_tokens[0] == q2_tokens[0])

- **abs_len_diff** : Abs. length difference
  abs_len_diff = abs(len(q1_tokens) - len(q2_tokens))

- **mean_len** : Average Token Length of both Questions
  mean_len = (len(q1_tokens) + len(q2_tokens))/2

- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-

fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage (https://github.com/seatgeek/fuzzywuzzy#usage) http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/ (http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/)

- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and Q2
  longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

In [10]:

```
def get_token_features(q1, q2):
    token_features = [0.0]*10

    # Converting the Sentence into Tokens:
    q1_tokens = q1.split()
    q2_tokens = q2.split()

    if len(q1_tokens) == 0 or len(q2_tokens) == 0:
        return token_features
    # Get the non-stopwords in Questions
    q1_words = set([word for word in q1_tokens if word not in STOP_WORDS])
    q2_words = set([word for word in q2_tokens if word not in STOP_WORDS])

    #Get the stopwords in Questions
    q1_stops = set([word for word in q1_tokens if word in STOP_WORDS])
    q2_stops = set([word for word in q2_tokens if word in STOP_WORDS])

    # Get the common non-stopwords from Question pair
    common_word_count = len(q1_words.intersection(q2_words))

    # Get the common stopwords from Question pair
    common_stop_count = len(q1_stops.intersection(q2_stops))

    # Get the common Tokens from Question pair
    common_token_count = len(set(q1_tokens).intersection(set(q2_tokens)))


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words))
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words))
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops))
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops))
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_toker
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_toker

    # Last word of both question is same or not
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

    # First word of both question is same or not
    token_features[7] = int(q1_tokens[0] == q2_tokens[0])

    token_features[8] = abs(len(q1_tokens) - len(q2_tokens))

    #Average Token Length of both Questions
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features

# get the Longest Common sub string

def get_longest_substr_ratio(a, b):
    strs = list(distance.lcsubstrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

def extract_features(df):
```

```
    # preprocessing each question
    df["question1"] = df["question1"].fillna("").apply(preprocess)
    df["question2"] = df["question2"].fillna("").apply(preprocess)

    print("token features...")

    # Merging Features with dataset

    token_features = df.apply(lambda x: get_token_features(x["question1"], x['

    df["cwc_min"]       = list(map(lambda x: x[0], token_features))
    df["cwc_max"]       = list(map(lambda x: x[1], token_features))
    df["csc_min"]       = list(map(lambda x: x[2], token_features))
    df["csc_max"]       = list(map(lambda x: x[3], token_features))
    df["ctc_min"]       = list(map(lambda x: x[4], token_features))
    df["ctc_max"]       = list(map(lambda x: x[5], token_features))
    df["last_word_eq"]  = list(map(lambda x: x[6], token_features))
    df["first_word_eq"] = list(map(lambda x: x[7], token_features))
    df["abs_len_diff"]  = list(map(lambda x: x[8], token_features))
    df["mean_len"]      = list(map(lambda x: x[9], token_features))

    #Computing Fuzzy Features and Merging with Dataset

    # do read this blog: http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string
    # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-func
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lambda x: fuzz.token_set_ratio(x["q
    # The token sort approach involves tokenizing the string in question, sort
    # then joining them back into a string We then compare the transformed str
    df["token_sort_ratio"]      = df.apply(lambda x: fuzz.token_sort_ratio(x['
    df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"
    df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["que
    df["longest_substr_ratio"]  = df.apply(lambda x: get_longest_substr_ratio(
    return df
```

In [12]:
```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[12]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | ... |
| **1** | 1 | 3 | 4 | what is the story of kohinoor koh i noor dia... | what would happen if the indian government sto... | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | ... |

2 rows × 21 columns

## 3.5.1 Analysis of extracted features

### 3.5.1.1 Plotting Word clouds

- Creating Word Cloud of Duplicates and Non-Duplicates Question pairs
- We can observe the most frequent occuring words

In [17]:
```
df_duplicate = df[df['is_duplicate'] == 1]
dfp_nonduplicate = df[df['is_duplicate'] == 0]

# Converting 2d array of q1 and q2 and flatten the array: like {{1,2},{3,4}} t
p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten(
n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).

print ("Number of data points in class 1 (duplicate pairs) :",len(p))
print ("Number of data points in class 0 (non duplicate pairs) :",len(n))

#Saving the np array into a text file
np.savetxt('train_p.txt', p, delimiter=' ', fmt='%s')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s')
```

```
Number of data points in class 1 (duplicate pairs) : 298526
Number of data points in class 0 (non duplicate pairs) : 510054
```

In [21]:

```
# reading the text files and removing the Stop Words:
d = path.dirname('.')

textp_w = open(path.join(d, 'train_p.txt')).read()
textn_w = open(path.join(d, 'train_n.txt')).read()
stopwords = set(STOPWORDS)
stopwords.add("said")
stopwords.add("br")
stopwords.add(" ")
stopwords.remove("not")

stopwords.remove("no")
#stopwords.remove("good")
#stopwords.remove("love")
stopwords.remove("like")
#stopwords.remove("best")
#stopwords.remove("!")
print ("Total number of words in duplicate pair questions :",len(textp_w))
print ("Total number of words in non duplicate pair questions :",len(textn_w))
```

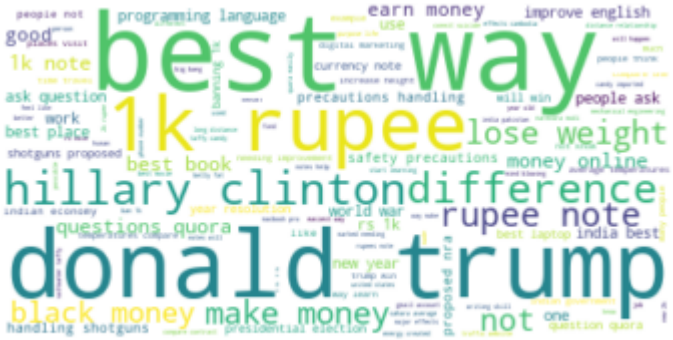
```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193130
```

__ Word Clouds generated from duplicate pair question's text __

In [22]:

```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stc
wc.generate(textp_w)
print ("Word Cloud for Duplicate Question pairs")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
Word Cloud for Duplicate Question pairs
```

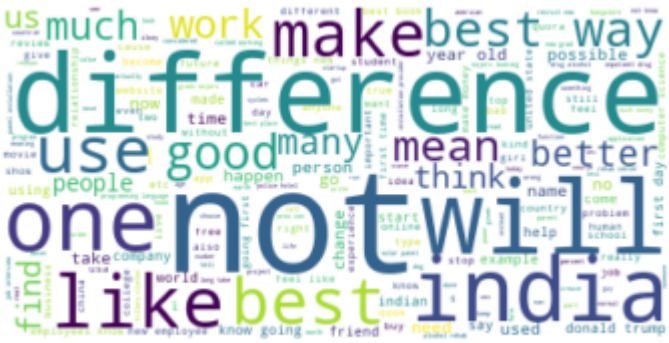



__ Word Clouds generated from non duplicate pair question's text __

In [23]:

```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stop
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

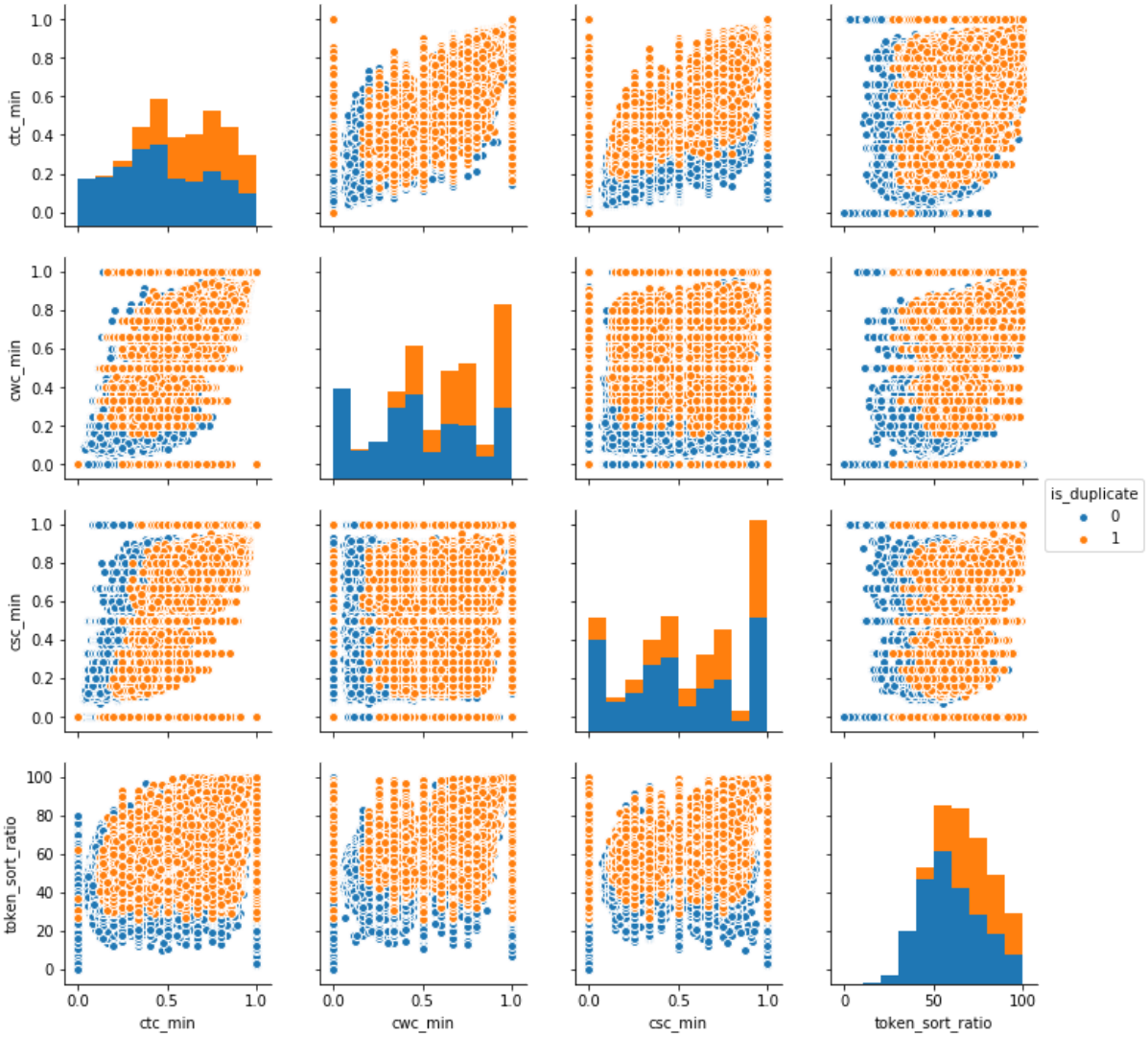
```
Word Cloud for non-Duplicate Question pairs:
```




### 3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

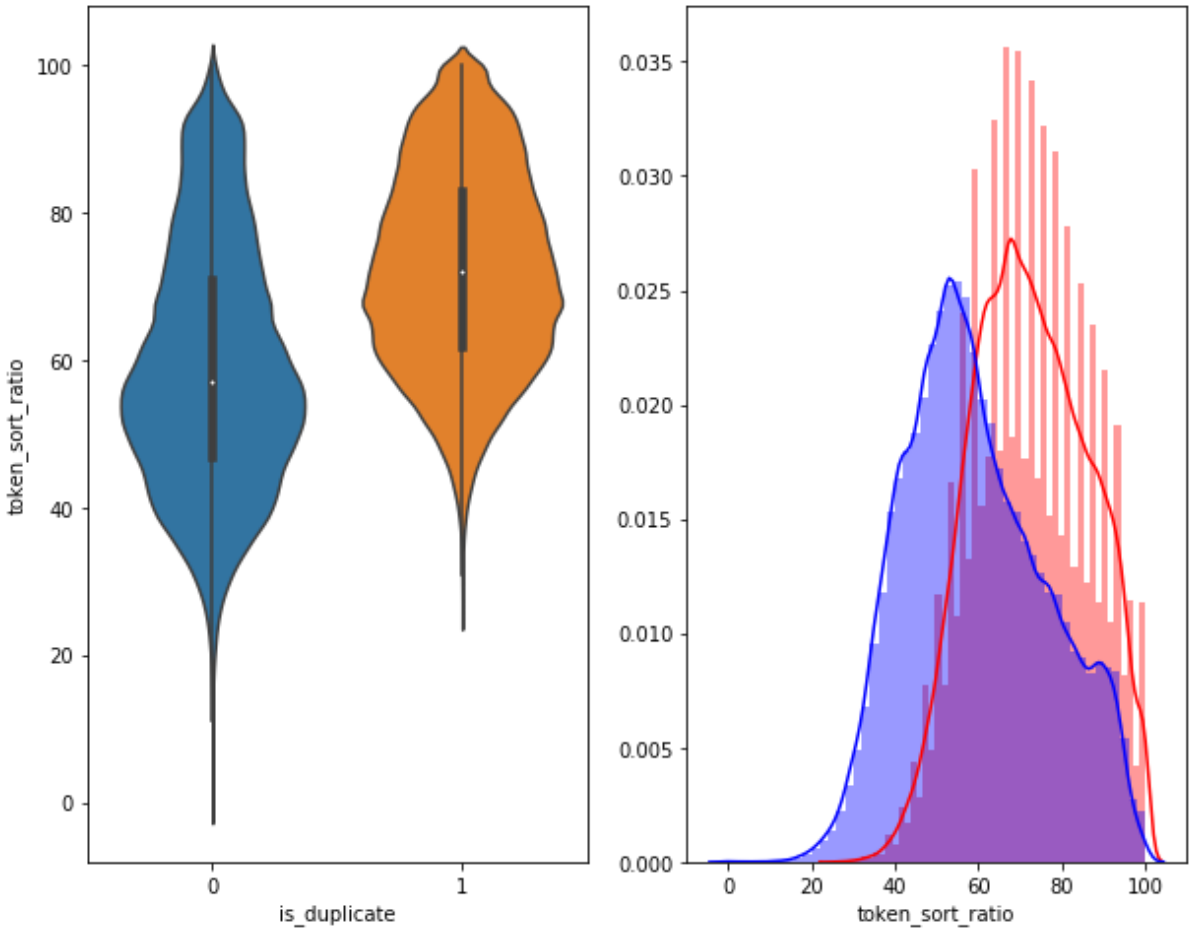
In [33]:

```
1 n = df.shape[0]
2 sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_dupl
3 plt.show()
```



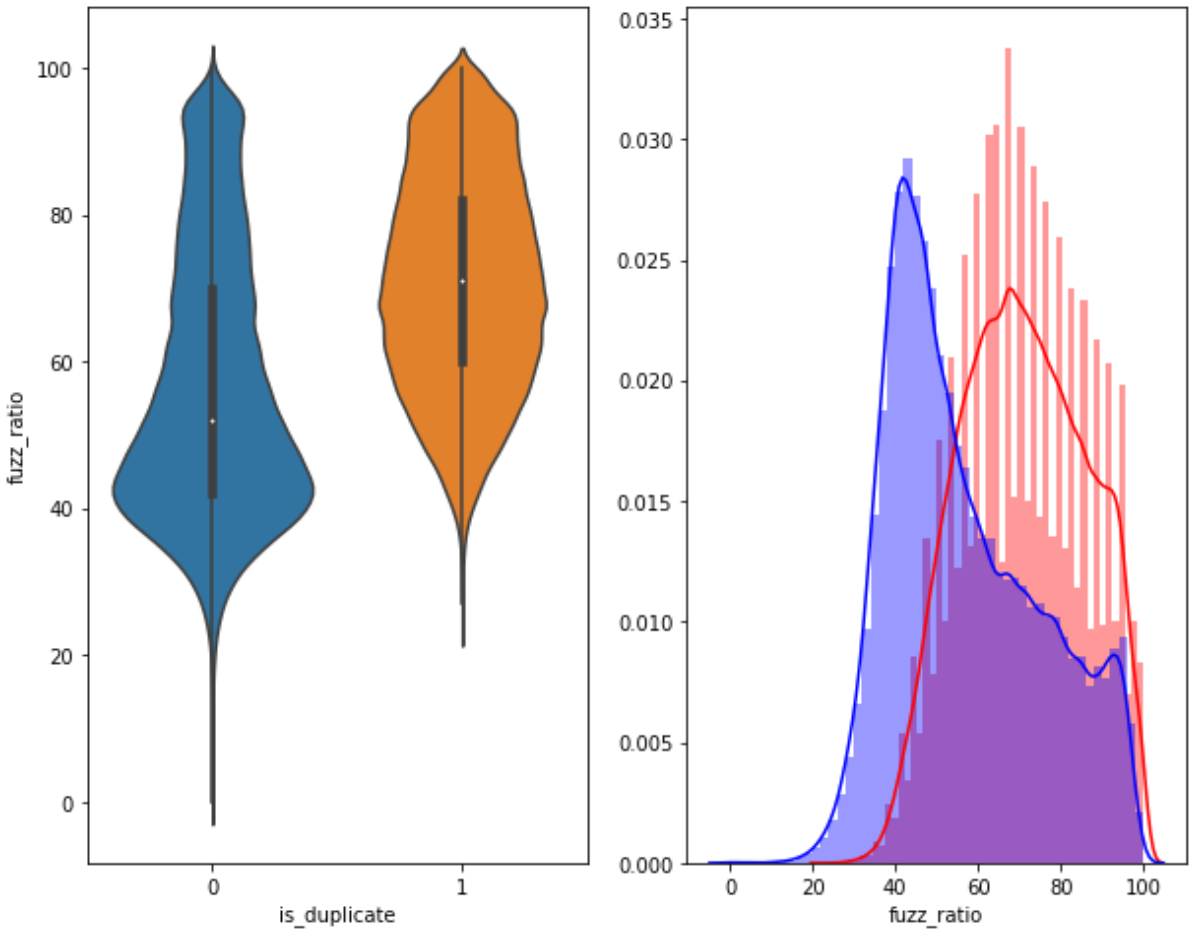
In [32]:

```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'token_sort_ratio', data = df[0:] , )

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['token_sort_ratio'][0:] , label = '
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = '
plt.show()
```



In [31]:

```
plt.figure(figsize=(10, 8))

plt.subplot(1,2,1)
sns.violinplot(x = 'is_duplicate', y = 'fuzz_ratio', data = df[0:] , )

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['fuzz_ratio'][0:] , label = "1", cc
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , c
plt.show()
```




### 3.5.2 Visualization

In [34]:

```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after clea

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_mi
y = dfp_subsampled['is_duplicate'].values
```

In [42]:

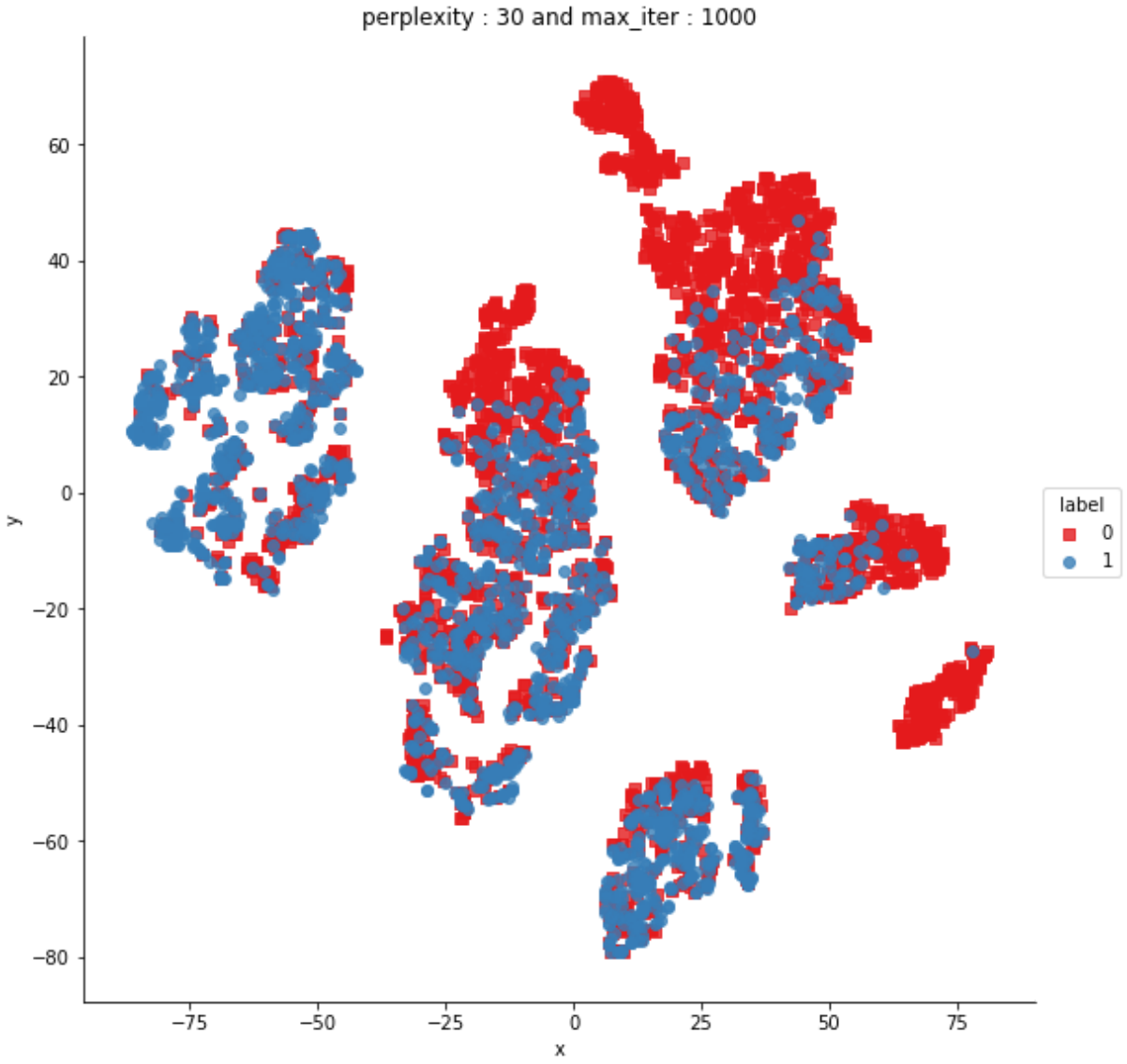
```
tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.011s...
[t-SNE] Computed neighbors for 5000 samples in 0.912s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.433s
[t-SNE] Iteration 50: error = 80.9244080, gradient norm = 0.0428133 (50 iterat
ions in 13.099s)
[t-SNE] Iteration 100: error = 70.3858795, gradient norm = 0.0100968 (50 itera
tions in 9.067s)
[t-SNE] Iteration 150: error = 68.6138382, gradient norm = 0.0058392 (50 itera
tions in 9.602s)
[t-SNE] Iteration 200: error = 67.7700119, gradient norm = 0.0036596 (50 itera
tions in 9.121s)
[t-SNE] Iteration 250: error = 67.2725067, gradient norm = 0.0034962 (50 itera
tions in 11.305s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.272507
[t-SNE] Iteration 300: error = 1.7737305, gradient norm = 0.0011918 (50 iterat
ions in 8.289s)
[t-SNE] Iteration 350: error = 1.3720417, gradient norm = 0.0004822 (50 iterat
ions in 10.526s)
[t-SNE] Iteration 400: error = 1.2039998, gradient norm = 0.0002768 (50 iterat
ions in 9.600s)
[t-SNE] Iteration 450: error = 1.1133438, gradient norm = 0.0001881 (50 iterat
ions in 11.827s)
[t-SNE] Iteration 500: error = 1.0579143, gradient norm = 0.0001434 (50 iterat
ions in 8.941s)
[t-SNE] Iteration 550: error = 1.0221983, gradient norm = 0.0001164 (50 iterat
ions in 11.092s)
[t-SNE] Iteration 600: error = 0.9987167, gradient norm = 0.0001039 (50 iterat
ions in 11.467s)
[t-SNE] Iteration 650: error = 0.9831534, gradient norm = 0.0000938 (50 iterat
ions in 11.799s)
[t-SNE] Iteration 700: error = 0.9722011, gradient norm = 0.0000858 (50 iterat
ions in 12.028s)
[t-SNE] Iteration 750: error = 0.9643636, gradient norm = 0.0000799 (50 iterat
ions in 12.120s)
[t-SNE] Iteration 800: error = 0.9584482, gradient norm = 0.0000785 (50 iterat
ions in 11.867s)
[t-SNE] Iteration 850: error = 0.9538348, gradient norm = 0.0000739 (50 iterat
ions in 11.461s)
[t-SNE] Iteration 900: error = 0.9496906, gradient norm = 0.0000712 (50 iterat
```

```
ions in 11.023s)
[t-SNE] Iteration 950: error = 0.9463405, gradient norm = 0.0000673 (50 iterat
ions in 11.755s)
[t-SNE] Iteration 1000: error = 0.9432716, gradient norm = 0.0000662 (50 itera
tions in 11.493s)
[t-SNE] Error after 1000 iterations: 0.943272
```

In [44]:

```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# draw the plot in appropriate place in the grid
sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,palette='
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



In [45]:

```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.010s...
[t-SNE] Computed neighbors for 5000 samples in 0.935s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.116557
[t-SNE] Computed conditional probabilities in 0.363s
[t-SNE] Iteration 50: error = 77.7944183, gradient norm = 0.1014017 (50 iterat
ions in 34.931s)
[t-SNE] Iteration 100: error = 69.2682266, gradient norm = 0.0248657 (50 itera
tions in 15.147s)
[t-SNE] Iteration 150: error = 67.7877655, gradient norm = 0.0150941 (50 itera
tions in 13.761s)
[t-SNE] Iteration 200: error = 67.1991119, gradient norm = 0.0126559 (50 itera
tions in 13.425s)
[t-SNE] Iteration 250: error = 66.8560715, gradient norm = 0.0074975 (50 itera
tions in 12.904s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.856071
[t-SNE] Iteration 300: error = 1.2356015, gradient norm = 0.0007033 (50 iterat
ions in 13.302s)
[t-SNE] Iteration 350: error = 0.9948602, gradient norm = 0.0001997 (50 iterat
ions in 18.898s)
[t-SNE] Iteration 400: error = 0.9168936, gradient norm = 0.0001430 (50 iterat
ions in 13.397s)
[t-SNE] Iteration 450: error = 0.8863022, gradient norm = 0.0000975 (50 iterat
ions in 16.379s)
[t-SNE] Iteration 500: error = 0.8681002, gradient norm = 0.0000854 (50 iterat
ions in 17.791s)
[t-SNE] Iteration 550: error = 0.8564141, gradient norm = 0.0000694 (50 iterat
ions in 17.060s)
[t-SNE] Iteration 600: error = 0.8470711, gradient norm = 0.0000640 (50 iterat
ions in 15.454s)
[t-SNE] Iteration 650: error = 0.8389117, gradient norm = 0.0000561 (50 iterat
ions in 17.562s)
[t-SNE] Iteration 700: error = 0.8325295, gradient norm = 0.0000529 (50 iterat
ions in 13.443s)
[t-SNE] Iteration 750: error = 0.8268463, gradient norm = 0.0000528 (50 iterat
ions in 17.981s)
[t-SNE] Iteration 800: error = 0.8219477, gradient norm = 0.0000477 (50 iterat
ions in 17.448s)
[t-SNE] Iteration 850: error = 0.8180174, gradient norm = 0.0000490 (50 iterat
ions in 18.376s)
```

```
[t-SNE] Iteration 900: error = 0.8150476, gradient norm = 0.0000456 (50 iterat
ions in 17.778s)
[t-SNE] Iteration 950: error = 0.8122067, gradient norm = 0.0000472 (50 iterat
ions in 16.983s)
[t-SNE] Iteration 1000: error = 0.8095787, gradient norm = 0.0000489 (50 itera
tions in 18.581s)
[t-SNE] Error after 1000 iterations: 0.809579
```

In [46]:

```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

data=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered feature
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

## Computing Word Vectors (Average W2V)

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]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
```

In [5]:
```
tfidf.fit_transform(questions)

```

Out[5]:
```
<808580x109679 sparse matrix of type '<class 'numpy.float64'>'
        with 8146554 stored elements in Compressed Sparse Row format>
```

In [4]:
```


# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

- 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 (https://spacy.io/usage/vectors-similarity)
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [5]:
```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 384])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df['q1_feats_m'] = list(vecs1)

```

```
100%|██████████████████████████████████████████████████████████████████████████████| 
404290/404290 [2:13:51<00:00, 50.34it/s]
```

In [6]:
```
vecs2 = []
for qu2 in tqdm(list(df['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 384])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df['q2_feats_m'] = list(vecs2)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
404290/404290 [1:47:52<00:00, 62.46it/s]
```

In [7]:
```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('nlp_features_train.csv'):
    dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous notebook

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='lat
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run pr
```

In [8]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)
df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
```

In [9]:
```
# dataframe of nlp features
df1.head()
```

Out[9]:

| | id | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ctc_min | ctc_max | last_word_eq | first |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | 0.916659 | 0.785709 | 0.0 | |
| **1** | 1 | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | 0.699993 | 0.466664 | 0.0 | |
| **2** | 2 | 0 | 0.399992 | 0.333328 | 0.399992 | 0.249997 | 0.399996 | 0.285712 | 0.0 | |
| **3** | 3 | 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | |
| **4** | 4 | 0 | 0.399992 | 0.199998 | 0.999950 | 0.666644 | 0.571420 | 0.307690 | 0.0 | |

In [10]:
```
# data before preprocessing
df2.head()
```

Out[10]:

| | id | freq_qid1 | freq_qid2 | q1len | q2len | q1_n_words | q2_n_words | word_Common | word_Total | w |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 1 | 66 | 57 | 14 | 12 | 10.0 | 23.0 | |
| **1** | 1 | 4 | 1 | 51 | 88 | 8 | 13 | 4.0 | 20.0 | |
| **2** | 2 | 1 | 1 | 73 | 59 | 14 | 10 | 4.0 | 24.0 | |
| **3** | 3 | 1 | 1 | 50 | 65 | 11 | 9 | 0.0 | 19.0 | |
| **4** | 4 | 3 | 1 | 76 | 39 | 13 | 7 | 2.0 | 20.0 | |

In [11]:
```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[11]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **0** | 121.929927 | 100.083900 | 72.497894 | 115.641800 | -48.370870 | 34.619058 | -172.057787 | -92.502617 |
| **1** | -78.070939 | 54.843781 | 82.738482 | 98.191872 | -51.234859 | 55.013510 | -39.140730 | -82.692352 |
| **2** | -5.355015 | 73.671810 | 14.376365 | 104.130241 | 1.433537 | 35.229116 | -148.519385 | -97.124595 |
| **3** | 5.778359 | -34.712038 | 48.999631 | 59.699204 | 40.661263 | -41.658731 | -36.808594 | 24.170655 |
| **4** | 51.138220 | 38.587312 | 123.639488 | 53.333041 | -47.062739 | 37.356212 | -298.722753 | -106.421119 |

5 rows × 384 columns

In [12]:
```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[12]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **0** | 125.983301 | 95.636485 | 42.114702 | 95.449980 | -37.386295 | 39.400078 | -148.116070 | -87.851475 |
| **1** | -106.871904 | 80.290331 | 79.066297 | 59.302092 | -42.175328 | 117.616655 | -144.364237 | -127.131513 |
| **2** | 7.072875 | 15.513378 | 1.846914 | 85.937583 | -33.808811 | 94.702337 | -122.256856 | -114.009530 |
| **3** | 39.421531 | 44.136989 | -24.010929 | 85.265863 | -0.339022 | -9.323137 | -60.499651 | -37.044763 |
| **4** | 31.950101 | 62.854106 | 1.778164 | 36.218768 | -45.130875 | 66.674880 | -106.342341 | -22.901008 |

5 rows × 384 columns

In [13]:
```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", df3_q1.shape[1])
print("Number of features in question2 w2v  dataframe :", df3_q2.shape[1])
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 384
Number of features in question2 w2v  dataframe : 384
Number of features in final dataframe  : 794
```

In [14]:
```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1['id']=df1['id']
    df3_q2['id']=df1['id']
    df1  = df1.merge(df2, on='id',how='left')
    df2  = df3_q1.merge(df3_q2, on='id',how='left')
    result  = df1.merge(df2, on='id',how='left')
    result.to_csv('final_features.csv')
```

# Applying Machine Learning Algorithms

### Reading data from file and storing into sql table

In [4]:
```
#Creating db file from csv
if not os.path.isfile('train.db'):
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('final_features.csv', names=['Unnamed: 0','id','is_d
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
```

In [5]:
```python
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None


def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the databse:")
    tables =table_names.fetchall()
    print(tables[0][0])
    return(len(tables))
```

In [6]:
```python
read_db = 'train.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

```
Tables in the databse:
data
```

In [7]:
```python
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", cor

        # for selecting random points
        data = pd.read_sql_query("SELECT * From data ORDER BY RANDOM() LIMIT 1
        conn_r.commit()
        conn_r.close()
```

In [8]:
```python
# remove the first row
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','index','is_duplicate'], axis=1, inplace=True)
```

In [9]:
```
1 data.head()
```

Out[9]:

| | cwc_min | cwc_max | csc_min | csc_max | ctc_mi |
|---|---|---|---|---|---|
| 1 | 0.199996000079998 | 0.166663888935184 | 0.0 | 0.0 | 0.1428551020699 |
| 2 | 0.399992000159997 | 0.399992000159997 | 0.499987500312492 | 0.499987500312492 | 0.44443950622770 |
| 3 | 0.833319444675922 | 0.714275510349852 | 0.999983333611106 | 0.857130612419823 | 0.68749570315185 |
| 4 | 0.0 | 0.0 | 0.599988000239995 | 0.499991666805553 | 0.24999791668402 |
| 5 | 0.749981250468738 | 0.749981250468738 | 0.499987500312492 | 0.499987500312492 | 0.62499218759765 |

5 rows × 794 columns

## Converting strings to numerics

In [10]:
```
1 # after we read from sql table each entry was read it as a string
2 # we convert all the features into numaric before we apply any model
3 cols = list(data.columns)
4 for i in cols:
5     data[i] = data[i].apply(pd.to_numeric)
6     print(i)
```

```
cwc_min
cwc_max
csc_min
csc_max
ctc_min
ctc_max
last_word_eq
first_word_eq
abs_len_diff
mean_len
token_set_ratio
token_sort_ratio
fuzz_ratio
fuzz_partial_ratio
longest_substr_ratio
freq_qid1
freq_qid2
q1len
q2len
```

In [18]:
```
1 # https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to
2 y_true = list(map(int, y_true.values))
```

## Random train test split( 70:30)

In [19]:
```
1 X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_tr
```

In [20]:
```
f
```

```
Number of data points in train data : (70000, 794)
Number of data points in test data : (30000, 794)
```

In [25]:
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6324857142857143 Class 1:  0.36751428571428574
---------- Distribution of output variable in train data ----------
Class 0:  0.3675 Class 1:  0.3675
```

In [34]:

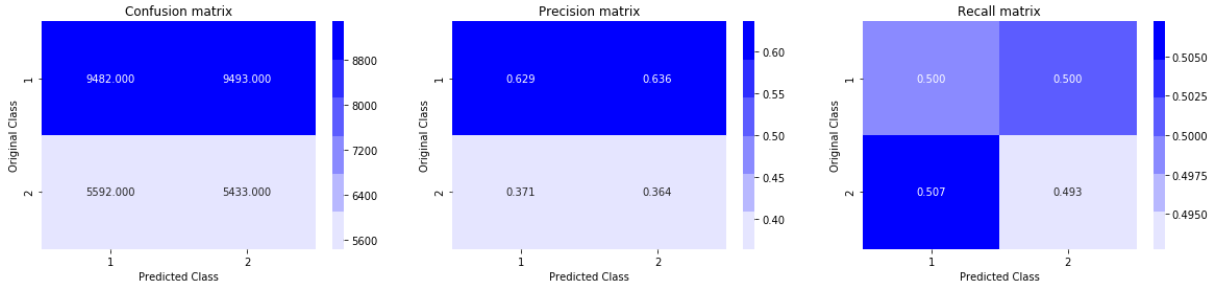
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i c

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

    # 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
    # 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 th
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytick
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytick
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytick
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

# Building a random model (Finding worst-case log-loss)

In [35]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y,

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

```
Log loss on Test Data using Random Model 0.887242646958
```



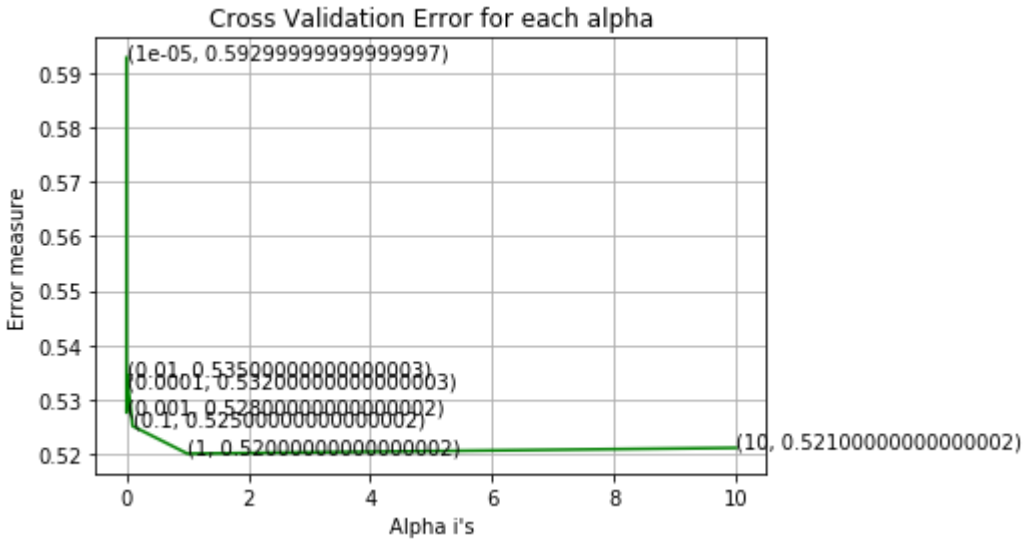

# Logistic Regression with hyperparameter tuning

In [50]:

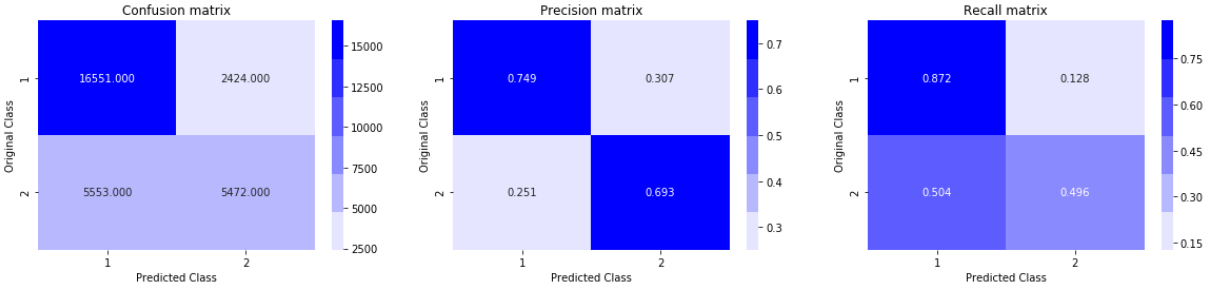
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
# 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 Stochastic G
# 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(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, ep
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre

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_
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
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: 0.592800211149
For values of alpha =  0.0001 The log loss is: 0.532351700629
```

```
For values of alpha =  0.001 The log loss is: 0.527562275995
For values of alpha =  0.01 The log loss is: 0.534535408885
For values of alpha =  0.1 The log loss is: 0.525117052926
For values of alpha =  1 The log loss is: 0.520035530431
For values of alpha =  10 The log loss is: 0.521097925307
```




```
For values of best alpha =  1 The train log loss is: 0.513842874233
For values of best alpha =  1 The test log loss is: 0.520035530431
Total number of data points : 30000
```



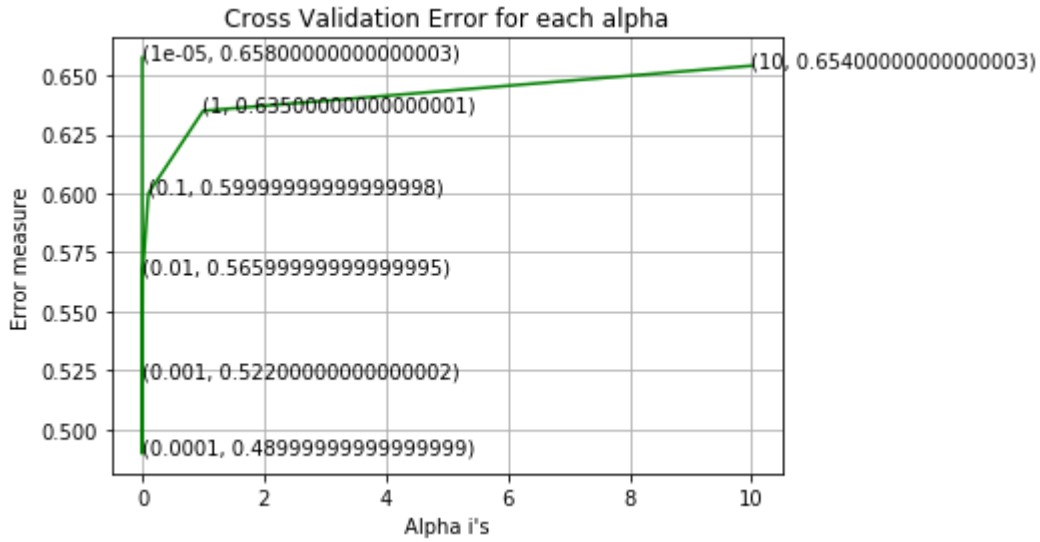

## Linear SVM with hyperparameter tuning

In [51]:

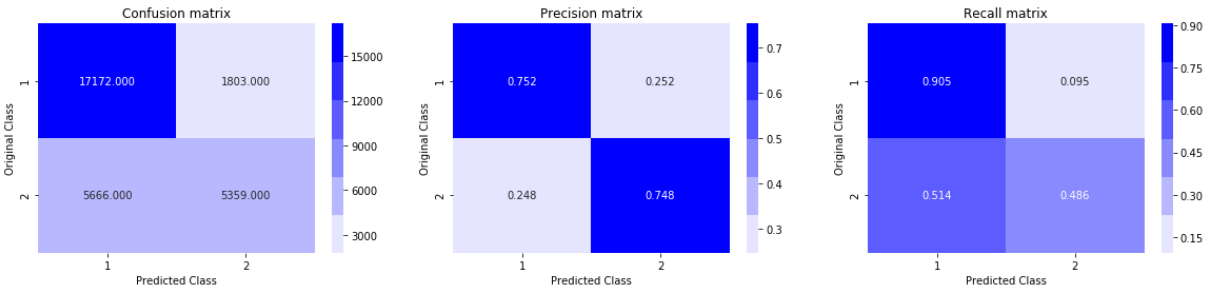
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
# 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 Stochastic G
# 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_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, ep
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre

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', rando
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
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: 0.657611721261
For values of alpha =  0.0001 The log loss is: 0.489669093534
```

```
For values of alpha =  0.001 The log loss is: 0.521829068562
For values of alpha =  0.01 The log loss is: 0.566295616914
For values of alpha =  0.1 The log loss is: 0.599957866217
For values of alpha =  1 The log loss is: 0.635059427016
For values of alpha =  10 The log loss is: 0.654159467907
```




```
For values of best alpha =  0.0001 The train log loss is: 0.478054677285
For values of best alpha =  0.0001 The test log loss is: 0.489669093534
Total number of data points : 30000
```




# XGBoost

In [52]:

```
import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4

d_train = xgb.DMatrix(X_train, label=y_train)
d_test = xgb.DMatrix(X_test, label=y_test)

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

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, ver

xgdmat = xgb.DMatrix(X_train,y_train)
predict_y = bst.predict(d_test)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,
```

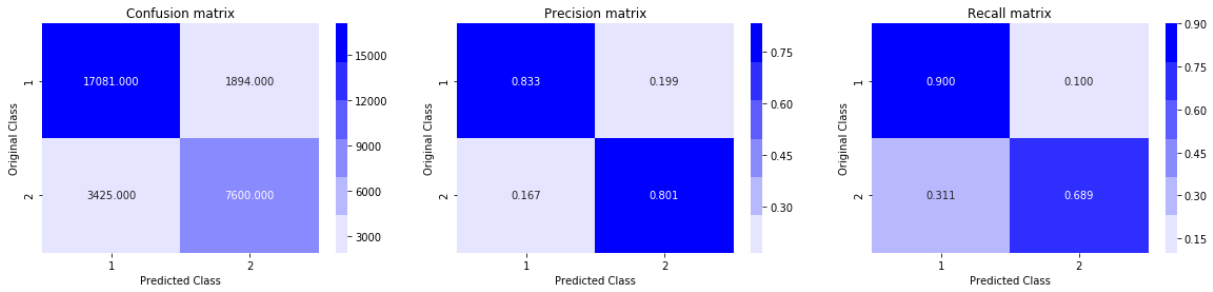
```
[0]     train-logloss:0.684819  valid-logloss:0.684845
Multiple eval metrics have been passed: 'valid-logloss' will be used for early
stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]    train-logloss:0.61583   valid-logloss:0.616104
[20]    train-logloss:0.564616  valid-logloss:0.565273
[30]    train-logloss:0.525758  valid-logloss:0.52679
[40]    train-logloss:0.496661  valid-logloss:0.498021
[50]    train-logloss:0.473563  valid-logloss:0.475182
[60]    train-logloss:0.455315  valid-logloss:0.457186
[70]    train-logloss:0.440442  valid-logloss:0.442482
[80]    train-logloss:0.428424  valid-logloss:0.430795
[90]    train-logloss:0.418803  valid-logloss:0.421447
[100]   train-logloss:0.41069   valid-logloss:0.413583
[110]   train-logloss:0.403831  valid-logloss:0.40693
[120]   train-logloss:0.398076  valid-logloss:0.401402
[130]   train-logloss:0.393305  valid-logloss:0.396851
[140]   train-logloss:0.38913   valid-logloss:0.392952
[150]   train-logloss:0.385469  valid-logloss:0.389521
[160]   train-logloss:0.382327  valid-logloss:0.386667
[170]   train-logloss:0.379541  valid-logloss:0.384148
[180]   train-logloss:0.377014  valid-logloss:0.381932
[190]   train-logloss:0.374687  valid-logloss:0.379883
[200]   train-logloss:0.372585  valid-logloss:0.378068
[210]   train-logloss:0.370615  valid-logloss:0.376367
[220]   train-logloss:0.368559  valid-logloss:0.374595
[230]   train-logloss:0.366545  valid-logloss:0.372847
[240]   train-logloss:0.364708  valid-logloss:0.371311
[250]   train-logloss:0.363021  valid-logloss:0.369886
[260]   train-logloss:0.36144   valid-logloss:0.368673
[270]   train-logloss:0.359899  valid-logloss:0.367421
[280]   train-logloss:0.358465  valid-logloss:0.366395
[290]   train-logloss:0.357128  valid-logloss:0.365361
[300]   train-logloss:0.355716  valid-logloss:0.364315
[310]   train-logloss:0.354425  valid-logloss:0.363403
[320]   train-logloss:0.353276  valid-logloss:0.362595
[330]   train-logloss:0.352084  valid-logloss:0.361823
```

```
[340]   train-logloss:0.351051  valid-logloss:0.361167
[350]   train-logloss:0.349867  valid-logloss:0.36043
[360]   train-logloss:0.348829  valid-logloss:0.359773
[370]   train-logloss:0.347689  valid-logloss:0.359019
[380]   train-logloss:0.346607  valid-logloss:0.358311
[390]   train-logloss:0.345568  valid-logloss:0.357674
The test log loss is: 0.357054433715
```

In [60]:

```
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 : 30000
```




# Applying Logistic regression and linear SVM on Simple TFIDF features

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.shape
```

Out[3]: (404290, 6)

In [25]:

```
final_df = pd.read_csv('final_features.csv')
```

In [26]:

```
final_col = list(final_df.columns)
```

In [29]:

```
final_fil_df = final_df[final_col[1:29]]
final_fil_df.to_csv('final_features_fil.csv', index=False)
```

In [30]:
```
final_fil_df.head()
```

Out[30]:

| | id | is_duplicate | cwc_min | cwc_max | csc_min | csc_max | ctc_min | ctc_max | last_word_eq | first |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0.999980 | 0.833319 | 0.999983 | 0.999983 | 0.916659 | 0.785709 | 0.0 | |
| **1** | 1 | 0 | 0.799984 | 0.399996 | 0.749981 | 0.599988 | 0.699993 | 0.466664 | 0.0 | |
| **2** | 2 | 0 | 0.399992 | 0.333328 | 0.399992 | 0.249997 | 0.399996 | 0.285712 | 0.0 | |
| **3** | 3 | 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | |
| **4** | 4 | 0 | 0.399992 | 0.199998 | 0.999950 | 0.666644 | 0.571420 | 0.307690 | 0.0 | |

5 rows × 28 columns

In [33]:
```
comb_df = df.merge(final_fil_df, on=["id"])
```

In [35]:
```
comb_df.drop(['is_duplicate_x'], axis=1, inplace=True)
```

In [37]:
```
comb_df.rename(index=str, columns={'is_duplicate_y':'is_duplicate'}, inplace=T
```

In [40]:
```
comb_df.to_csv('comb_fil_df.csv', index=False)
```

In [42]:
```
comb_df.shape
```

Out[42]:
```
(404290, 32)
```

** Computing on all the data points is raising "MemoryError". So, I have used 10000 points from data. **

In [44]:
```
sam_comb_df = comb_df[:10000]

sam_comb_df.to_csv('sample_comb_fil_df.csv', index=False)
```

In [103]:
```
sam_comb_df = pd.read_csv('sample_comb_fil_df.csv', encoding='cp1252')
```

In [45]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(sam_comb_df['question1']) + list(sam_comb_df['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
```

In [46]:
```
tfidf_vec = tfidf.fit_transform(questions)

```

In [49]:
```
# Assigning the Sparse Matrices of Q1 and Q2 Sparse Matrices
q1_vec = tfidf_vec[:sam_comb_df.shape[0]]
q2_vec = tfidf_vec[sam_comb_df.shape[0]:]
```

In [55]:
```
q1_vec = q1_vec.toarray()
q2_vec = q2_vec.toarray()
```

In [58]:
```
q1_vec.shape
```

Out[58]: (10000, 17418)

In [61]:
```
index_x = ["x_" + str(i) for i in range(0, q1_vec.shape[1]) ]
index_y = ["y_" + str(i) for i in range(0, q2_vec.shape[1])]
df3_q1 = pd.DataFrame(data=q1_vec, columns=index_x)
df3_q2 = pd.DataFrame(data=q2_vec, columns=index_y)
```

** Making X and Y datasets. **

In [104]:
```
sam_comb_df_features = sam_comb_df.iloc[:,6:]
```

In [92]:
```
sam_tfidf_features  = pd.concat([df3_q1, df3_q2], axis=1)
```

In [93]:
```
sam_tfidf_features.shape
```

Out[93]: (10000, 34836)

In [94]:
```
sam_comb_df_features.shape
```

Out[94]: (10000, 26)

In [105]:
```
sam_tfidf_features_arr = sam_tfidf_features.values
sam_com_df_features_arr = sam_comb_df_features.values
```

In [109]:
```
com_data = np.hstack((sam_tfidf_features_arr, sam_com_df_features_arr))
```

In [111]:
```
np.save('com_data.npy', com_data)
```

In [85]:
```
y = sam_comb_df['is_duplicate'].values
```

In [121]:
```
np.save('com_data_y.npy',y)
```

** Train and test splitting of data. **

In [87]:
```
from sklearn.model_selection import train_test_split
```

In [112]:

```
1 X_train,X_test, y_train, y_test = train_test_split(com_data, y, stratify=y, t
```

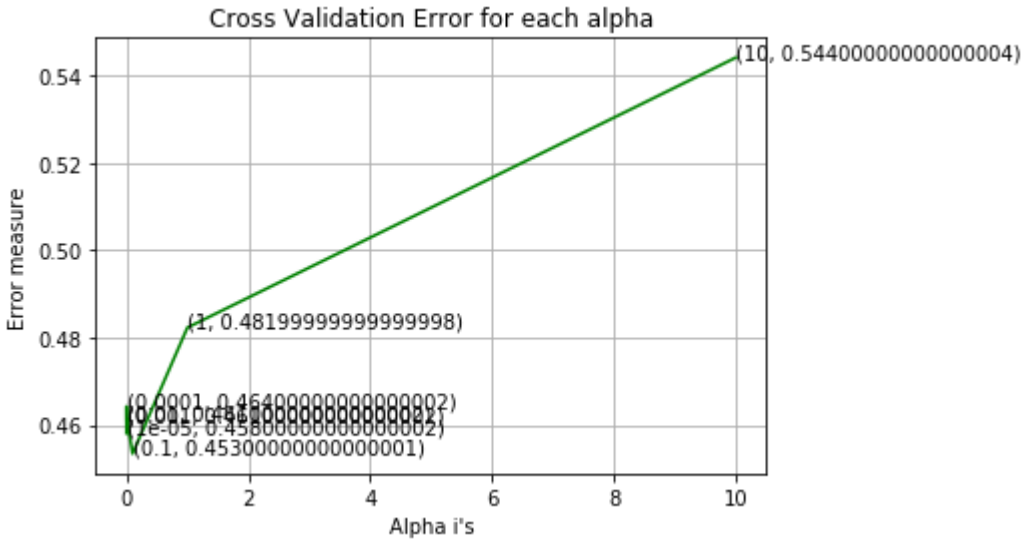
** Logistic Regression. **

In [118]:

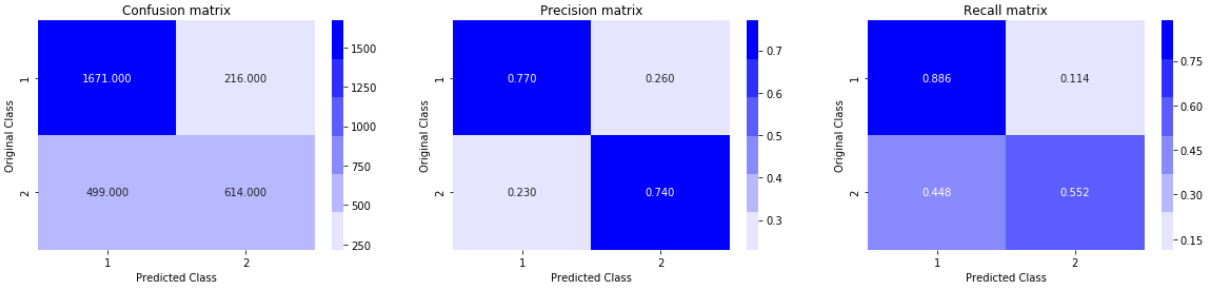
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    print("For Alpha: {} \n".format(i))
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, e
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pr

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
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
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 Alpha: 1e-05

For values of alpha =  1e-05 The log loss is: 0.458121867813
For Alpha: 0.0001

For values of alpha =  0.0001 The log loss is: 0.464271808076
For Alpha: 0.001

For values of alpha =  0.001 The log loss is: 0.46098815888
For Alpha: 0.01

For values of alpha =  0.01 The log loss is: 0.461200475162
For Alpha: 0.1

For values of alpha =  0.1 The log loss is: 0.453438911955
For Alpha: 1
```

```
For values of alpha =  1 The log loss is: 0.482321302379
For Alpha: 10

For values of alpha =  10 The log loss is: 0.544186641282
```




```
For values of best alpha =  0.1 The train log loss is: 0.469279314863
For values of best alpha =  0.1 The test log loss is: 0.453438911955
Total number of data points : 3000
```



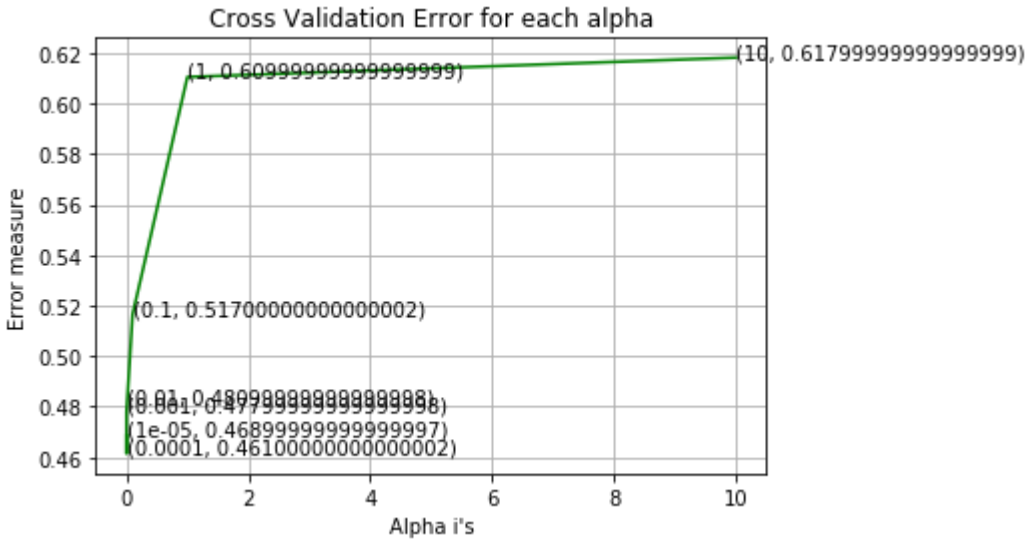

** Support Vector Machine.**

In [120]:

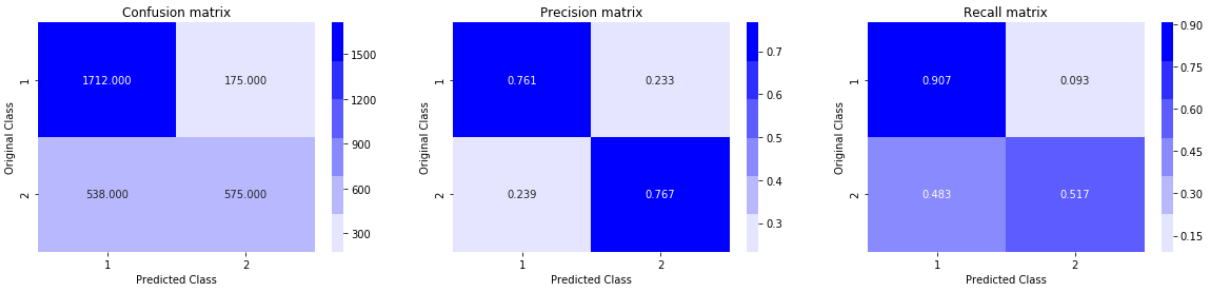
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    print("For Alpha: {} \n".format(i))
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, e
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pr

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', rand
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
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 Alpha: 1e-05

For values of alpha =  1e-05 The log loss is: 0.468595574651
For Alpha: 0.0001

For values of alpha =  0.0001 The log loss is: 0.461435202833
For Alpha: 0.001

For values of alpha =  0.001 The log loss is: 0.478022693837
For Alpha: 0.01

For values of alpha =  0.01 The log loss is: 0.481395005323
For Alpha: 0.1

For values of alpha =  0.1 The log loss is: 0.516529462333
For Alpha: 1
```

```
For values of alpha =  1 The log loss is: 0.610460609251
For Alpha: 10

For values of alpha =  10 The log loss is: 0.618137656884
```




```
For values of best alpha =  0.0001 The train log loss is: 0.47891025282
For values of best alpha =  0.0001 The test log loss is: 0.461435202833
Total number of data points : 3000
```




# XGBoost Hyper Parameter Tuning

In [63]:

```python
def xgboost(x_train, x_test, y_train, y_test):
    alpha = {
        'learning_rate' : [0.1, 0.2, 0.3],
     'n_estimators':[i*10 for i in range(1, 10)],
     'max_depth':[i for i in range(1,5)],
    }

    x_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logl
    xgb_model = GridSearchCV(estimator=x_model, param_grid=alpha, cv=10, scori
    xgb_model.fit(x_train, y_train)
    print('The Best Parameters are: \n')
    print(xgb_model.best_params_)
    predict_y = xgb_model.predict(x_test)
    print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
    plot_confusion_matrix(y_test, predict_y)
    return None
```

** The model is fitted on the sample of the data usin 10000 data points. **

In [64]:
```
sam_df = pd.read_csv('sample_comb_fil_df.csv', encoding='cp1252')
```

In [65]:
```
data = sam_df.drop(['is_duplicate', 'question1', 'question2', 'id'], axis=1)
y = sam_df['is_duplicate'].values
```

In [66]:
```
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.33, s
```

In [67]:
```
X_train.shape
```

Out[67]:
```
(6700, 28)
```

In [68]:
```
X_train.head()
```

Out[68]:

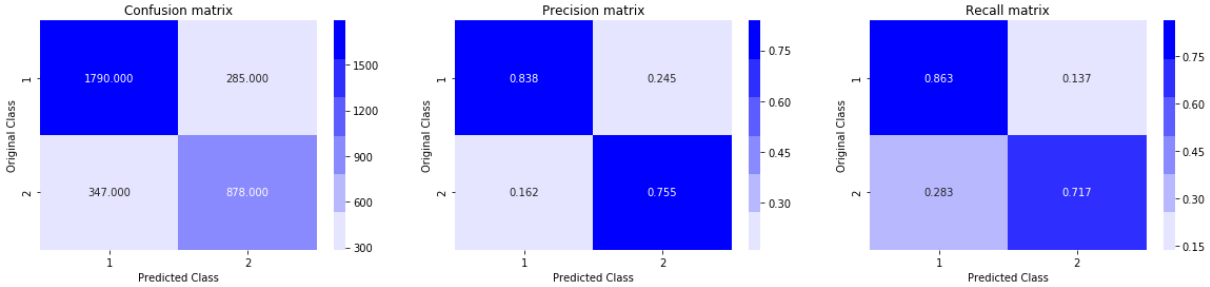
| | qid1 | qid2 | cwc_min | cwc_max | csc_min | csc_max | ctc_min | ctc_max | last_word_eq | firs |
|---|---|---|---|---|---|---|---|---|---|---|
| **5883** | 11552 | 11553 | 0.000000 | 0.000000 | 0.499994 | 0.444440 | 0.210525 | 0.210525 | 0.0 | |
| **2889** | 5729 | 5730 | 0.749981 | 0.374995 | 0.666644 | 0.499988 | 0.714276 | 0.416663 | 0.0 | |
| **6069** | 11901 | 11902 | 0.499988 | 0.333328 | 0.799984 | 0.799984 | 0.599994 | 0.545450 | 1.0 | |
| **6392** | 12526 | 12527 | 0.999950 | 0.666644 | 0.000000 | 0.000000 | 0.499988 | 0.499988 | 1.0 | |
| **66** | 133 | 134 | 0.499988 | 0.499988 | 0.999967 | 0.499992 | 0.714276 | 0.499995 | 0.0 | |

5 rows × 28 columns

## 4.6 XGBoost

In [69]:
```
1 xgboost(X_train, X_test, y_train, y_test)
```

```
Fitting 10 folds for each of 108 candidates, totalling 1080 fits

[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:   15.1s
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:   16.0s
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:   17.2s
[Parallel(n_jobs=-1)]: Done  24 tasks      | elapsed:   18.5s
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:   20.6s
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   22.7s
[Parallel(n_jobs=-1)]: Done  53 tasks      | elapsed:   25.6s
[Parallel(n_jobs=-1)]: Done  64 tasks      | elapsed:   28.6s
[Parallel(n_jobs=-1)]: Done  77 tasks      | elapsed:   33.0s
[Parallel(n_jobs=-1)]: Done  90 tasks      | elapsed:   38.0s
[Parallel(n_jobs=-1)]: Done 105 tasks      | elapsed:   40.8s
[Parallel(n_jobs=-1)]: Done 120 tasks      | elapsed:   44.6s
[Parallel(n_jobs=-1)]: Done 137 tasks      | elapsed:   50.4s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:   57.7s
[Parallel(n_jobs=-1)]: Done 173 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 213 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 234 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 257 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 305 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done 330 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 384 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 413 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 473 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 504 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 537 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 570 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 605 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done 677 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done 714 tasks      | elapsed:  5.5min
[Parallel(n_jobs=-1)]: Done 753 tasks      | elapsed:  5.6min
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done 833 tasks      | elapsed:  5.9min
[Parallel(n_jobs=-1)]: Done 874 tasks      | elapsed:  6.2min
[Parallel(n_jobs=-1)]: Done 917 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done 960 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:  7.2min
[Parallel(n_jobs=-1)]: Done 1050 tasks      | elapsed:  7.6min
[Parallel(n_jobs=-1)]: Done 1080 out of 1080 | elapsed:  8.0min finished

The Best Parameters are:

{'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 90}
The test log loss is: 6.61476805056
```




**The value of the log_loss is more as the model training is done on the sample data.**

In [ ]:
```
1
```