# 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.

### 1.3 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

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

### 2.2.2 Performance Metric

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

Metric(s):

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

# 1. Business Problem

In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=
email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2f
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%
2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleap
i.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [2]:

```
pip install git+git://github.com/seatgeek/fuzzywuzzy.git@0.17.0#egg=fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Cloning git://github.com/seatgeek/fuzzywuzzy.git (to revision 0.17.0) to /
tmp/pip-install-gdej9nyx/fuzzywuzzy
  Running command git clone -q git://github.com/seatgeek/fuzzywuzzy.git /tm
p/pip-install-gdej9nyx/fuzzywuzzy
  Running command git checkout -q 089e5c46a117e3de293fd6cb94580efc2b9f6912
Building wheels for collected packages: fuzzywuzzy
  Building wheel for fuzzywuzzy (setup.py) ... done
  Created wheel for fuzzywuzzy: filename=fuzzywuzzy-0.17.0-py2.py3-none-any.
whl size=18151 sha256=85b2fe8b0959d0ae85d04e142bdbdfd8c1c3246924399f413826a2
081e55d1b6
  Stored in directory: /tmp/pip-ephem-wheel-cache-3zul206g/wheels/93/cb/79/4
e772f8c8772e2f67a1548610943f8b23b2a83a278206bc5d9
Successfully built fuzzywuzzy
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

In [3]:

```
'''!python -m spacy download en_core_web_lg'''
```

Out[3]:

```
'!python -m spacy download en_core_web_lg'
```

In [4]:

```
pip install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437ae
fa0d0a92ccfb38895d9416bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz (180kB)
     |████████████████████████████████| 184kB 2.8MB/s
Building wheels for collected packages: distance
  Building wheel for distance (setup.py) ... done
  Created wheel for distance: filename=Distance-0.1.3-cp36-none-any.whl size
=16261 sha256=92ad110a8e5347b74b531f890094a18119235757d8c6387c16a4fc3e07e9da
e5
  Stored in directory: /root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d0f
12db1c66dbae9c5442b39b001db18e
Successfully built distance
Installing collected packages: distance
Successfully installed distance-0.1.3
```

In [5]:

```
pip install python-Levenshtein
```

```
Collecting python-Levenshtein
  Downloading https://files.pythonhosted.org/packages/42/a9/d1785c85ebf9b7df
acd08938dd028209c34a0ea3b1bcdb895208bd40a67d/python-Levenshtein-0.12.0.tar.g
z (48kB)
     |████████████████████████████████| 51kB 1.6MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-p
ackages (from python-Levenshtein) (41.4.0)
Building wheels for collected packages: python-Levenshtein
  Building wheel for python-Levenshtein (setup.py) ... done
  Created wheel for python-Levenshtein: filename=python_Levenshtein-0.12.0-c
p36-cp36m-linux_x86_64.whl size=144669 sha256=44b93a21b0c30ac6a7d4371dcd3fc3
e6ed6c3cfa94704e097f3a98b41514d2a2
  Stored in directory: /root/.cache/pip/wheels/de/c2/93/660fd5f7559049268ad2
dc6d81c4e39e9e36518766eaf7e342
Successfully built python-Levenshtein
Installing collected packages: python-Levenshtein
Successfully installed python-Levenshtein-0.12.0
```

In [6]:

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import nltk
nltk.download('stem')
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
import distance
from fuzzywuzzy import process
import os
from scipy.sparse import hstack as hs

# for text featurization
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import spacy
from sklearn.model_selection import train_test_split

from numpy import hstack

# model
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from collections import Counter
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, RandomizedSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score

from sklearn.model_selection import cross_val_score
from mlxtend.classifier import StackingClassifier
from xgboost import XGBClassifier

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

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
import matplotlib.patches as mpatches




import warnings
warnings.filterwarnings("ignore")
```

```
[nltk_data] Error loading stem: Package 'stem' not found in index
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: Deprecat
ionWarning: The module is deprecated in version 0.21 and will be removed in
version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

In [0]:

```
data = pd.read_csv('/content/drive/My Drive/Quora/train.csv')
data = data.sample(100000)
```

# 3. Exploratory Data Analysis

In [161]:

```
data.columns
```

Out[161]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate'], dtyp
e='object')
```

In [162]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 254740 to 232193
Data columns (total 6 columns):
id              100000 non-null int64
qid1            100000 non-null int64
qid2            100000 non-null int64
question1       100000 non-null object
question2       100000 non-null object
is_duplicate    100000 non-null int64
dtypes: int64(4), object(2)
```

```
memory usage: 5.3+ MB
```

In [163]:

```
data.head()
```

Out[163]:

|  | id | qid1 | qid2 | question1 | question2 | is_duplicate |
|---|---|---|---|---|---|---|
| **254740** | 254740 | 369525 | 369526 | How can I stop myself thinking about my ex who... | How do I stop myself from contacting/thinking ... | 1 |
| **341718** | 341718 | 418224 | 294841 | How can I make ice cream project? | How do you make ice cream? | 0 |
| **293090** | 293090 | 48667 | 278028 | What is best website for learning? | What is the best website for online learning? | 1 |
| **165871** | 165871 | 83026 | 86107 | Why did MS Dhoni give up captaincy? | Why MS Dhoni has quit the captaincy from limit... | 1 |
| **183808** | 183808 | 280976 | 280977 | What builds work well in Protoss vs Zerg in Le... | StarCraft II: What are some of the easiest way... | 0 |

## Distribution of data points among output classes

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

In [164]:

```
total = data.shape[0]
duplicate = data[data.is_duplicate==1].shape[0]
not_duplicate = data[data.is_duplicate==0].shape[0]

print("similar question pairs {:.2f}%".format((duplicate/total)*100))
print("non similar question pairs {:.2f}%".format((not_duplicate/total)*100))
```

```
similar question pairs 37.09%
non similar question pairs 62.91%
```

Above we can see that only 36.92% of question are duplicate and 63.08% percent are not duplicate. that means there are more number of non duplicate questions.

In [165]:

```
data.groupby('is_duplicate')['id'].count().plot.bar()
plt.title('bar graph representing duplicate and non duplicate questions')
```

Out[165]:

```
Text(0.5, 1.0, 'bar graph representing duplicate and non duplicate question
s')
```

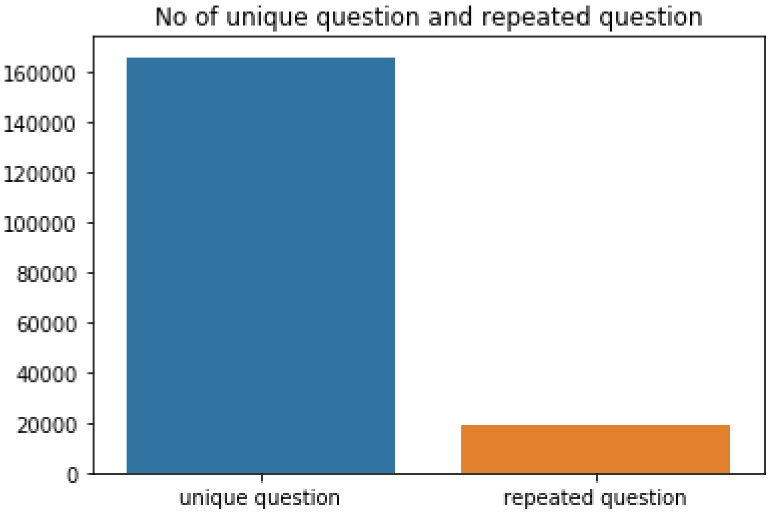
bar graph representing duplicate and non duplicate questions

## Number of unique questions

```
In [166]: questions = pd.Series(data['qid1'].tolist() + data['qid2'].tolist())
          unique = len(np.unique(questions))
          more_than_1 = np.sum(questions.value_counts()>1)
          print("total number of unique questions are = ", unique)
          print("total number of questions appearing more than once = ", more_than_1)
```

```
total number of unique questions are =  166153
total number of questions appearing more than once =  19316
```

```
In [167]: sns.barplot(['unique question', 'repeated question'], [unique, more_than_1])
          plt.title('No of unique question and repeated question')
          plt.show()
```



# checking for null values

```
In [168]: # now looking for null values in the data
          print("is there null value in question1= {} , {} questions are null".format(dat
          a['question1'].isnull().any(),data['question1'].isnull().sum() ))
          print("is there null value in question2= {}, {} questions are null".format(data
          ['question2'].isnull().any(), data['question2'].isnull().sum()))
```

```
is there null value in question1= False , 0 questions are null
is there null value in question2= False, 0 questions are null
```

```
In [0]: # we can see that both question1 and question2 can have null values
        # filling null values with ''
        data['question1'].fillna(' ', inplace=True)
        data['question2'].fillna(' ', inplace=True)
```

```
In [170]: print("is there null value in question1= {} , {} questions are null".format(dat
          a['question1'].isnull().any(),data['question1'].isnull().sum() ))
          print("is there null value in question2= {}, {} questions are null".format(data
          ['question2'].isnull().any(), data['question2'].isnull().sum()))
```

```
is there null value in question1= False , 0 questions are null
is there null value in question2= False, 0 questions are null
```

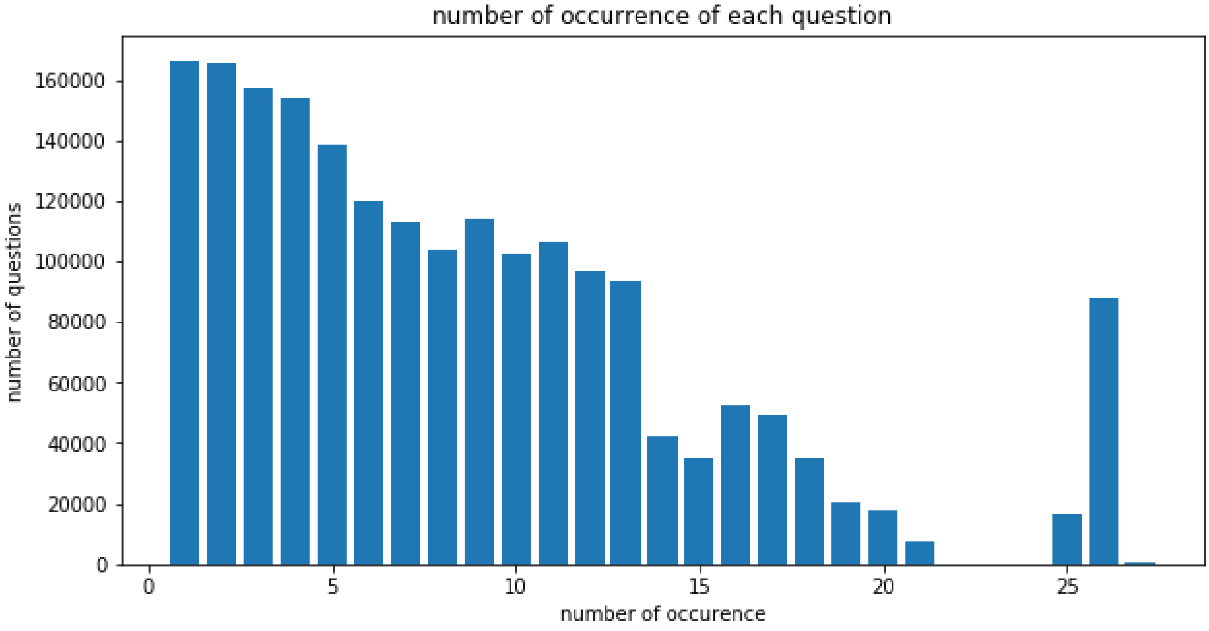
## Number of occurrences of each question

In [0]:

```
# getting the number of occurence of each question
qids = []
qids.extend(data['qid1'].tolist())
qids.extend(data['qid2'].tolist())

from collections import Counter

occurence = Counter(qids)
```

In [172]:

```
%%time
plt.figure(figsize=(10,5))
plt.bar(list(occurence.values()),range(len(occurence)), align='center')
plt.xlabel('number of occurence')
plt.ylabel('number of questions')
plt.title('number of occurrence of each question')
plt.show()
plt.close()
```



```
CPU times: user 5min 53s, sys: 2.27 s, total: 5min 55s
Wall time: 5min 55s
```

In [0]:

```
# getting frequency of each questions
q1 = Counter(data['qid1'].tolist())
q2 = Counter(data['qid2'].tolist())

data['freq_1'] = [q1[i] for i in data['qid1'].tolist() ]
data['freq_2'] = [q2[i] for i in data['qid2'].tolist() ]
```

In [174]:

```
 # removing stopwords from the question
 # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from tqdm import tqdm


stop_words = set(stopwords.words('english'))

filtered_isduplicate1 = []
filtered_isduplicate2 = []
filtered_notduplicate1 = []
filtered_notduplicate2 = []
```

```
for i, j in tqdm(zip(data[data.is_duplicate==0]['question1'].tolist(), data[dat
a.is_duplicate==0]['question2'].tolist())):
    temp  = ''
    for w in i.split():
        if w not in stop_words:
            temp = temp + w + " "
    filtered_notduplicate1.append(temp)

    temp  = ''
    for w in j.split():
        if w not in stop_words:
            temp = temp + w + " "
    filtered_notduplicate2.append(temp)

for i, j in tqdm(zip(data[data.is_duplicate==1]['question1'].tolist(), data[dat
a.is_duplicate==1]['question2'].tolist())):
    temp  = ''
    for w in i.split():
        if w not in stop_words:
            temp = temp + w + " "
    filtered_isduplicate1.append(temp)

    temp  = ''
    for w in j.split():
        if w not in stop_words:
            temp = temp + w + " "
    filtered_isduplicate2.append(temp)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
62910it [00:00, 120167.49it/s]
37090it [00:00, 138099.01it/s]
```

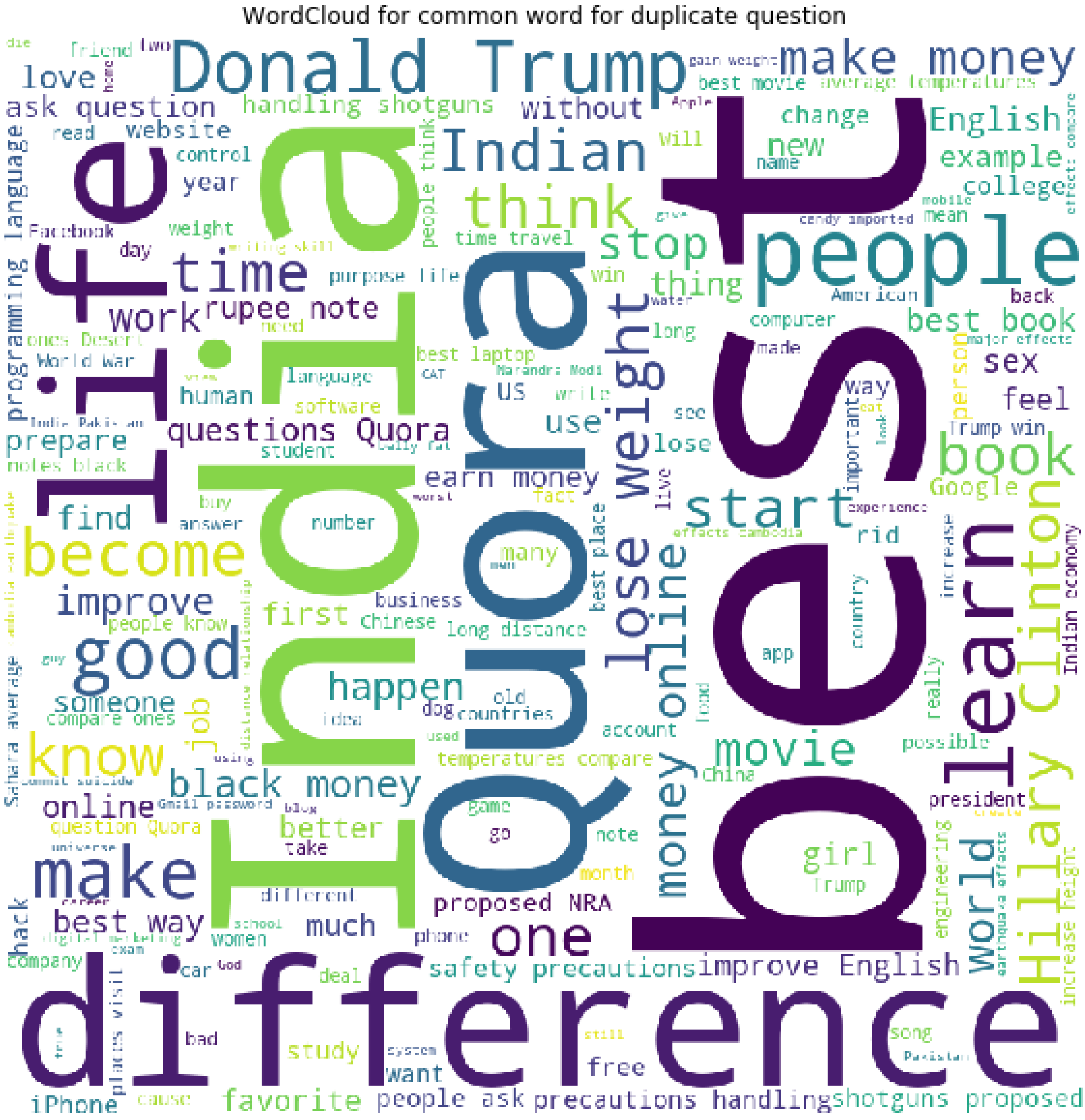
## Word cloud for common words

In [0]:

```
# getting common words list from the questions

common_words_isduplicate = ''
common_words_notduplicate = ''
for i, j in zip(filtered_isduplicate1, filtered_isduplicate2):
    for w in i.split():
        if w in j:
            common_words_isduplicate +=  w + " "

for i, j in zip(filtered_notduplicate1, filtered_notduplicate2):
    for w in i.split():
        if w in j:
            common_words_notduplicate +=  w + " "
```

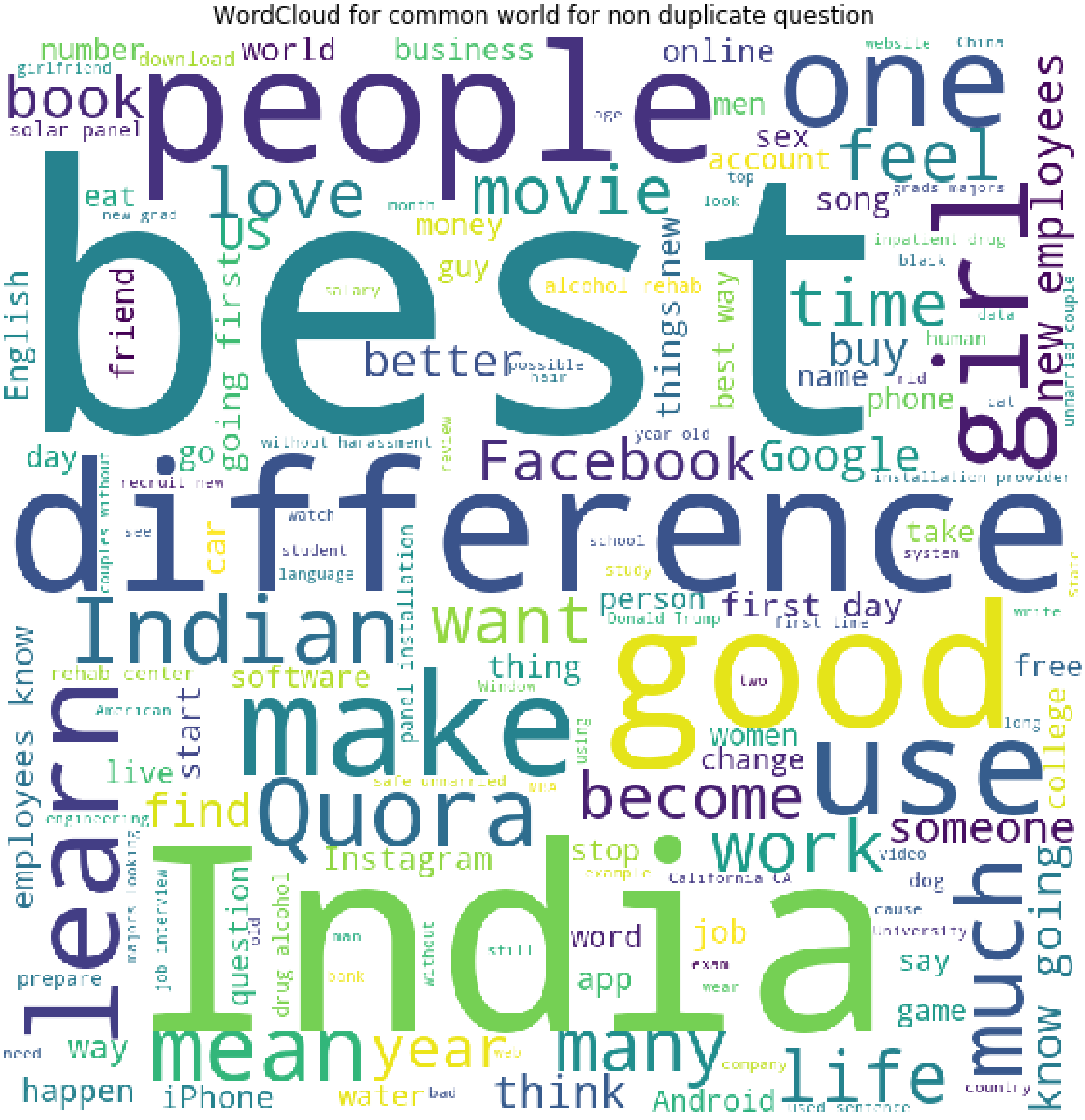
In [176]:

```
# generating word cloud for similar questions

# https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud, STOPWORDS

stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(common_words_isduplicate)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.title("WordCloud for common word for duplicate question")
plt.tight_layout(pad = 0)

plt.show()
```



## Word cloud for disimilar texts

In [177]:
```
# generating word cloud for disimilar question
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                     min_font_size = 10).generate(common_words_notduplicate)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.title("WordCloud for common world for non duplicate question")
plt.tight_layout(pad = 0)

plt.show()
```



In [178]:
```
data.head()
```

Out[178]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_1 | freq_ |
|---|---|---|---|---|---|---|---|---|
| **254740** | 254740 | 369525 | 369526 | How can I stop myself thinking about my ex who... | How do I stop myself from contacting/thinking ... | 1 | 1 | 1 |
| **341718** | 341718 | 418224 | 294841 | How can I make ice | How do you make ice cream? | 0 | 1 | 1 |

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  |  |  |  | cream project? |  |  |  |  |
| **293090** | 293090 | 48667 | 278028 | What is best website for learning? | What is the best website for online learning? | 1 | 1 | 2 |
| **165871** | 165871 | 83026 | 86107 | Why did MS Dhoni give up captaincy? | Why MS Dhoni has quit the captaincy from limit... | 1 | 2 | 5 |
| **183808** | 183808 | 280976 | 280977 | What builds work well in Protoss vs Zerg in Le... | StarCraft II: What are some of the easiest way... | 0 | 1 | 1 |

### Preprocessing the data

In [0]:

```
def preprocess(text):
    '''
    function to preprocess each question
    It includes:-
       1. Removing html tags
       2. removing puntuations
       3. removing stemming
       4. removing stopwords
    '''
    # removing html tags using beautifulsoup4
    text = text.lower()
    text = BeautifulSoup(text)
    text = text.get_text()

    # removing punctuations for the questions
    # https://stackoverflow.com/a/16799238/11746488

    text = re.sub(r'[^\w\s0-9]',' ',text)

    # removing stemming words and stop words

    pattern  = re.compile('\W')
    text = re.sub(pattern, ' ', text)

    stemming = PorterStemmer()
    temp_text = ''
    for word in text.split():
        if (word not in stop_words) and (type(word) == type('')):
            temp_text += stemming.stem(word) + ' '
    return temp_text.strip(' ')
```

In [180]:

```
# getting preprocessed text in a list
question1 = []
question2 = []
for i in tqdm(data['question1'].tolist()):
    question1.append(preprocess(i))
for i in tqdm(data['question2'].tolist()):
    question2.append(preprocess(i))
```

```
100%|██████████| 100000/100000 [00:45<00:00, 2210.72it/s]
100%|██████████| 100000/100000 [00:45<00:00, 2190.72it/s]
```

In [0]:
```
def extract_features(text1, text2, i, a):
    '''
    funtion to extract additional feature
    it includes:- 1. token count, 2. counting common token ratio, 3. counting stopword ratio
                  4. common word count, 5. avg token length, 6. absolute length difference,
                  7. common first words, 8. fuzzy ratio, 9. fuzzy token set,
                  10. fuzzy token sort, 11. fuzzy partial ratio, 12, longest common sequence

    '''

    token1 = text1.lower().split()
    token2 = text2.lower().split()

    # counting common token ratio
    # https://www.geeksforgeeks.org/python-intersection-of-two-string/
    # using set() + intersection() to
    # get string intersection
    ctc = len(set(token1).intersection(token2))
    if token1>token2:
        try:
            a[0][i] = (float(ctc/len(token2)))
        except ZeroDivisionError:
            a[0][i] = 0

        try:
            a[1][i] = (float(ctc/len(token1)))
        except ZeroDivisionError:
            a[1][i] = 0

    else:
        try:
            a[0][i] = (float(ctc/len(token1)))
        except ZeroDivisionError:
            a[0][i] = 0
        try:
            a[1][i] =(float(ctc/len(token2)))
        except ZeroDivisionError:
            a[1][i] = 0

    # common stop count
    q1 = [w for w in text1.split() if w in stop_words]
    q2 = [w for w in text2.split() if w in stop_words]

    csw = len(set(q1).intersection(q2))
    if len(q1)>len(q2):
        try:
            a[2][i] = (float(csw/len(q2)))
        except ZeroDivisionError:
            a[2][i] = 0
        try:
            a[3][i] = (float(csw/len(q1)))
        except ZeroDivisionError:
            a[3][i] = 0
    else:
        try:
            a[2][i] = (float(csw/len(q1)))
        except ZeroDivisionError:
            a[2][i]=0
        try:
            a[3][i] =(float(csw/len(q2)))
        except ZeroDivisionError:
            a[3][i] = 0

    # common word count
    q1 = [w for w in text1.split() if w not in stop_words]
    q2 = [w for w in text2.split() if w not in stop_words]

    cwc = len(set(q1).intersection(q2))
    if len(q1)>len(q2):
        try:
            a[4][i] = (float(cwc/len(q2)))
        except ZeroDivisionError:
            a[4][i] = 0
        try:
            a[5][i] = (float(cwc/len(q1)))
        except ZeroDivisionError:
            a[5][i] = 0
    else:
        try:
            a[4][i] = (float(cwc/len(q1)))
        except ZeroDivisionError:
            a[4][i]=0
        try:
            a[5][i] = (float(cwc/len(q2)))
        except ZeroDivisionError:
            a[5][i]=0

    # comparing common first word
    try:
        a[6][i] = (int( text1.split()[0]== text2.split()[0]))
    except:
        a[6][i] = 0

    # average token length
    a[7][i] = ((len(token1)+len(token2))/2)

    # absolute length difference b/w two questions
    a[8][i] = (abs(len(token1) - len(token2)))

    # levenstein distance
    ''' In information theory, linguistics and computer science, the Levenshtein distance is a string metric for measuring the difference between
    two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions
    or substitutions) required to change one word into the other.'''
    #  fuzzy ratio
    a[9][i] = (fuzz.ratio(text1.lower(), text2.lower()))

    # fuzzy partial ratio
    a[10][i] = (fuzz.partial_ratio(text1.lower(), text2.lower()))

    # fuzzy token sort ratio
    a[11][i] = (fuzz.token_sort_ratio(text1.lower(), text2.lower()))

    # fuzzy token set ratio
    a[12][i] = (fuzz.token_set_ratio(text1.lower(), text2.lower()))

    # longest substring ratio
    lcs = list(distance.lcsubstrings(text1, text2))
    if len(lcs) == 0:
        a[13][i] = (0)
    else:
        a[13][i] = (len(lcs)/(min(len(token1), len(token2))+1))

    return a
```

In [0]:

```
def add_features():

    a = np.zeros((14,len(data)))

    text1 = data['question1'].tolist()
    text2 = data['question2'].tolist()

    for i in tqdm(range(len(data))):
        a = extract_features(text1[i], text2[i], i, a)
        i+=1

    data['ctc_min'] = a[0]
    data['ctc_max'] = a[1]
    data['csw_min']  = a[2]
    data['csw_max']  = a[3]
    data['cwc_min']  = a[4]
    data['cwc_max']  = a[5]
    data['first_word'] = a[6]
    data['avg_token_length'] = a[7]
    data['abs_lnth_diff'] = a[8]
    data['fzy_ratio'] = a[9]
    data['fzy_partial_ratio'] = a[10]
    data['fzy_token_set'] = a[11]
    data['fzy_token_sort'] = a[12]
    data['lcsr'] = a[13]
```

In [183]:

```
add_features()
```

```
100%|██████████| 100000/100000 [01:31<00:00, 1095.31it/s]
```

In [0]:

```
# adding preprocessed text to data set
data['preprocessed_q1'] = question1
data['preprocessed_q2'] = question2
```

In [185]:

```
data.head()
```

Out[185]:

| | id | qid1 | qid2 | question1 | question2 | is_duplicate | freq_1 | freq_ |
|---|---|---|---|---|---|---|---|---|
| **254740** | 254740 | 369525 | 369526 | How can I stop myself thinking about my ex who... | How do I stop myself from contacting/thinking ... | 1 | 1 | 1 |
| **341718** | 341718 | 418224 | 294841 | How can I make ice cream project? | How do you make ice cream? | 0 | 1 | 1 |
| **293090** | 293090 | 48667 | 278028 | What is best website for learning? | What is the best website for online learning? | 1 | 1 | 2 |
| **165871** | 165871 | 83026 | 86107 | Why did MS Dhoni | Why MS Dhoni has quit the | 1 | 2 | 5 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | give up captaincy? | captaincy from limit... | | | |
| **183808** | 183808 | 280976 | 280977 | What builds work well in Protoss vs Zerg in Le... | StarCraft II: What are some of the easiest way... | 0 | 1 | 1 |

In [0]:

```
y = data['is_duplicate']
x = data.drop(['is_duplicate', 'question1', 'question2'], axis=1)
Xtrain, Xtest, Ytrain, Ytest = train_test_split(x, y, test_size=0.33)
```

In [248]:

```
count_ = list(Ytrain).count(0)
print("distribution of non duplicate class ",count_/len(Ytrain))
print("distribution of duplicate class ",(len(Ytrain)-count_)/len(Ytrain))
```

```
distribution of non duplicate class  0.6291492537313433
distribution of duplicate class  0.3708507462686567
```

In [0]:

```
# featurizing text data with tfidf-word vector
questions = list(Xtrain['preprocessed_q1']) + list(Xtrain['preprocessed_q1'])
tfidf = TfidfVectorizer()
tfidf.fit(questions)
a = (tfidf.transform(Xtrain['preprocessed_q1']))
b = (tfidf.transform(Xtrain['preprocessed_q2']))

c = tfidf.transform(Xtest['preprocessed_q1'])
d = tfidf.transform(Xtest['preprocessed_q2'])

word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [0]:

```
def w2vec(questions, word2tfidf):
    nlp = spacy.load('en_core_web_sm')
    vector = []
    for q in tqdm(questions):
        doc = nlp(q) # getting word representation of the question
        try:
            mean_vector = np.zeros((len(doc), len(doc[0].vector)))
        except:
            doc = nlp(' ')
            mean_vector = np.zeros((len(doc), len(doc[0].vector)))
            vector.append(mean_vector.mean(axis=0))
            continue
        for word in doc:
            vect  = word.vector # getting the vector representation for each wo
rd
            # gettin idf values of each word from tfidf vector
            try:
                idf = word2tfidf[str(word)]
            except:
                idf = 0
            mean_vector += vect * idf # multiplyiing vector representation of t
he word with its idf values
        mean_vector = mean_vector.mean(axis=0)
        vector.append(mean_vector)
    return vector
```

In [251]:

```
# getting w2vec vector for train data
```

```
a1= w2vec(list(Xtrain['preprocessed_q1']), word2tfidf)

a2= w2vec(list(Xtrain['preprocessed_q2']), word2tfidf)
```

```
100%|██████████| 67000/67000 [09:53<00:00, 112.81it/s]
100%|██████████| 67000/67000 [09:55<00:00, 112.56it/s]
```

In [252]:

```
# getting word to vec for test data
b1 = w2vec(list(Xtest['preprocessed_q1']), word2tfidf)

b2 = w2vec(list(Xtest['preprocessed_q2']), word2tfidf)
```

```
100%|██████████| 33000/33000 [04:54<00:00, 112.06it/s]
100%|██████████| 33000/33000 [04:55<00:00, 111.55it/s]
```

In [0]:

```
Xtrain.drop(['preprocessed_q1', 'preprocessed_q2'], axis=1, inplace=True)
Xtest.drop(['preprocessed_q1', 'preprocessed_q2'], axis=1, inplace=True)
```

In [0]:

```
n = np.asarray(Xtrain.values)
n1 = np.asarray(Xtest.values)
```

In [0]:

```
# stacking train and test data
train = hs((a,b,n)) #  data with tfidf vector
test = hs((c,d,n1))

train_2 = hstack((np.asarray(a1), np.asarray(a2), n)) #  data with tfidf-w2vec
 vector
test_2 = hstack((np.asarray(b1), np.asarray(b2), n1))
```

In [0]:

```
def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        predictions.append(1 if i>=threshold else 0)
    return predictions
```

In [0]:

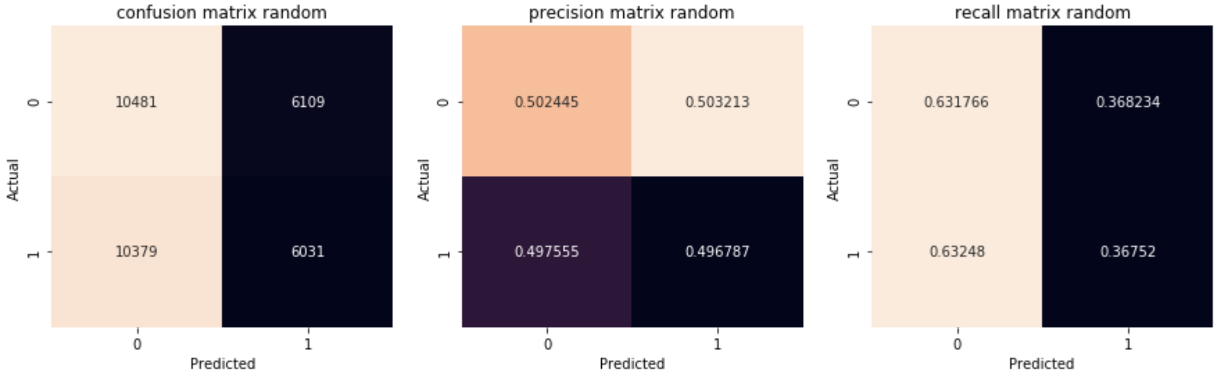
```
def precision(arr):
    arr = arr.astype(float)
    for i in range(2):
        x = arr[0][i]+arr[1][i]
        arr[0][i] = arr[0][i]/x
        arr[1][i] = arr[1][i]/x
    return arr

def recall(arr):
    arr = arr.astype(float)
    for i in range(2):
        x = arr[i][0]+arr[i][1]
        arr[i][0] = float(arr[i][0]/x)
        arr[i][1] = float(arr[i][1]/x)
    return arr

def confussion_matrix(pred, actual, t ):

    # https://datatofish.com/confusion-matrix-python/
    d = {'y_Predicted': list(pred),
          'y_Actual':    actual
          }
    df = pd.DataFrame(d, columns=['y_Actual','y_Predicted'])
    a = pd.crosstab(df['y_Actual'], df['y_Predicted'], rownames=['Actual'], col
names=['Predicted'])

    p = precision(np.array(a))
    p = pd.DataFrame(p, columns=['0', '1'])

    r = recall(np.array(a))
    r = pd.DataFrame(r, columns=['0', '1'])

    fig = plt.figure(figsize=(15,4))

    plt.subplot(1, 3, 1)
    ax = sns.heatmap(a,cbar=False, fmt='g', annot=True)
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    plt.title('confusion matrix '+t)

    plt.subplot(1, 3, 2)
    df_corr = p.corr()
    ax = sns.heatmap(p,cbar=False, fmt='g', annot=True)
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    plt.title('precision matrix '+t)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')

    plt.subplot(1, 3, 3)
    ax = sns.heatmap(r,cbar=False, fmt='g', annot=True)
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    plt.title('recall matrix '+t)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
    plt.close()
```

#### Random Model

In [0]:

```
# 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 their
sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
t = len(Ytest)
predicted_y = np.zeros((t,2))
for i in range(t):
    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(Ytest, predicted_y, e
ps=1e-15))

predicted_y =np.argmax(predicted_y, axis=1)
confussion_matrix(Ytest, predicted_y, 'random')
```

```
Log loss on Test Data using Random Model 0.886734072621072
```



# Summary

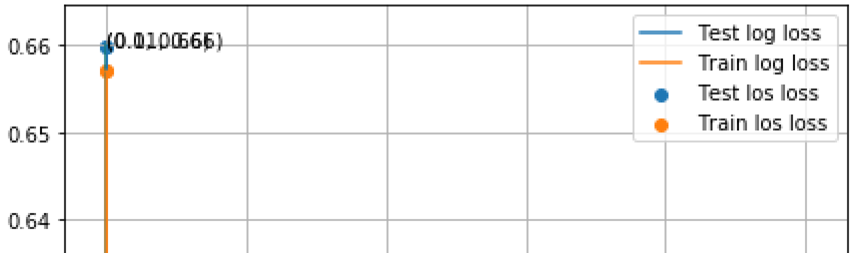
We can see that if we use the random model for our data we get the log loss of 0.887 which is very poor

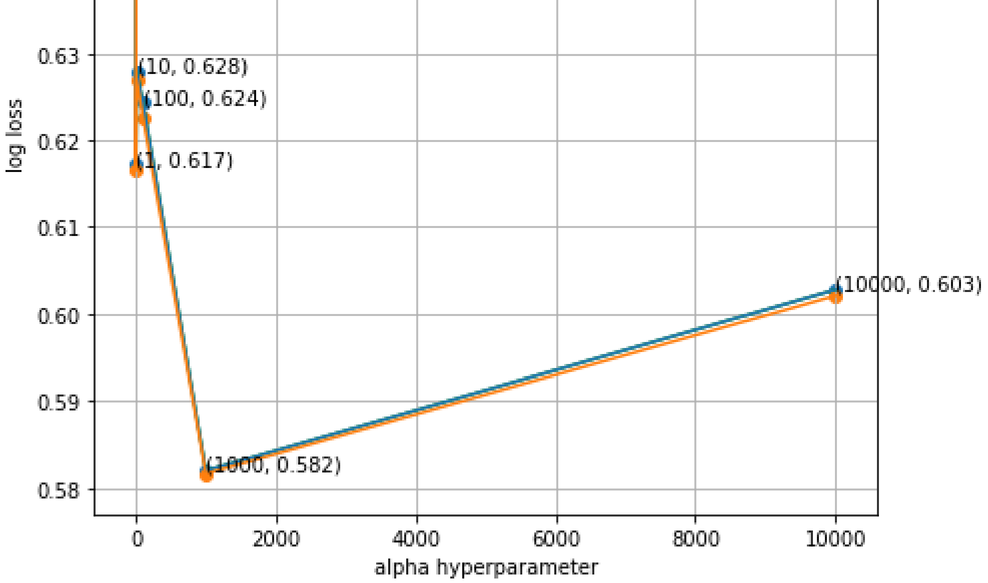
### Logistic Regression

In [0]:

```
%%time
parameter = [10**x for x in range(-2,5)]
log_error_test = []
log_error_train = []

for i in (parameter):
    LR = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42) # using SGDClassifier for logistic regression with loss= log
    sig_clf = CalibratedClassifierCV(LR, method="sigmoid")
    sig_clf.fit(train, Ytrain)
    predicted_train = sig_clf.predict_proba(train)
    predicted_test = sig_clf.predict_proba(test)
    log_error_test.append(log_loss(Ytest, predicted_test, labels=sig_clf.classes_, eps=1e-15))
    log_error_train.append(log_loss(Ytrain, predicted_train, labels=sig_clf.classes_, eps=1e-15))
    print(' For values of alpha = ', i, "The log loss is:",log_loss(Ytest, predicted_test, labels=sig_clf.classes_, eps=1e-15))

plt.figure(figsize = (7, 7), facecolor = None)
plt.plot(parameter, log_error_test,c='g')
for i, txt in enumerate(np.round(log_error_test,3)):
    plt.annotate((parameter[i],np.round(txt,3)), (parameter[i],log_error_test[i]))

plt.plot(parameter, log_error_test, label='Test log loss')
plt.scatter(parameter, log_error_test, label='Test los loss')
plt.plot(parameter, log_error_train, label='Train log loss')
plt.scatter(parameter, log_error_train, label='Train los loss')

plt.title('Hyper parameter v/s log loss')
plt.xlabel('alpha hyperparameter')
plt.ylabel('log loss')
plt.title('')
plt.grid()
plt.legend()

plt.show()

plt.close()
```

```
 For values of alpha =  0.01 The log loss is: 0.659962416173107
 For values of alpha =  0.1 The log loss is: 0.659962416173107
 For values of alpha =  1 The log loss is: 0.6172207779075802
 For values of alpha =  10 The log loss is: 0.6279525438922822
 For values of alpha =  100 The log loss is: 0.6242902603811931
 For values of alpha =  1000 The log loss is: 0.5820093471693567
 For values of alpha =  10000 The log loss is: 0.6027720635127032
```



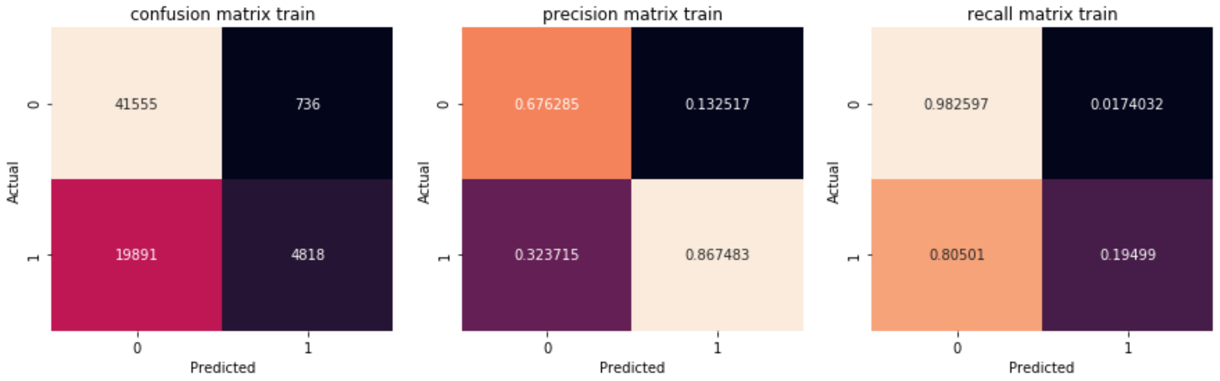


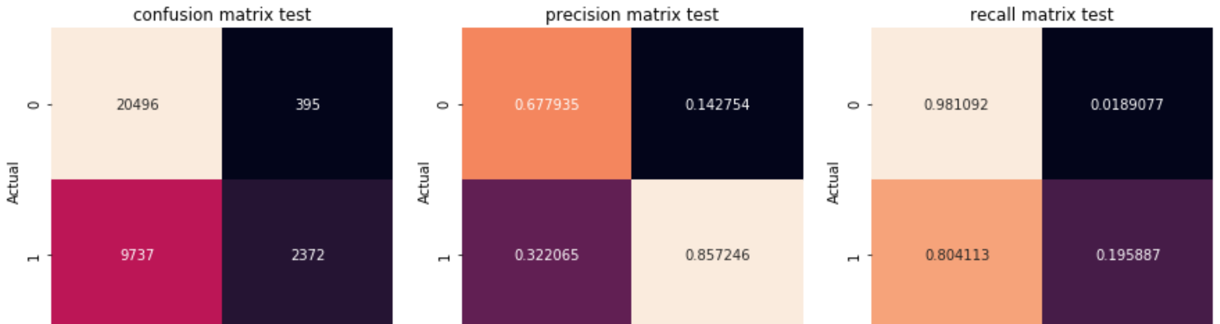
```
CPU times: user 1min 16s, sys: 4.26 s, total: 1min 20s
Wall time: 1min 15s
```

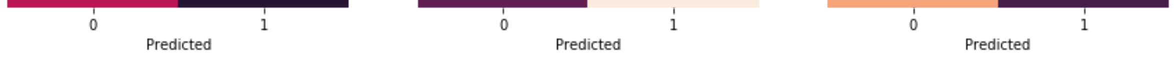
In [0]:
```
best_alpha = np.argmin(log_error_test) # choosing alpha with minimum log loss
LR = SGDClassifier(alpha=parameter[best_alpha], penalty='l2', loss='log', rando
m_state=42) # using SGDClassifier for logistic regression with loss= log
sig_clf = CalibratedClassifierCV(LR, method="sigmoid")
sig_clf.fit(train, Ytrain)
predicted_train = sig_clf.predict_proba(train)
predicted_train = [i[1] for i in predicted_train ]
predicted_test = sig_clf.predict_proba(test)
predicted_test = [i[1] for i in predicted_test]

print(' For alpha = ', parameter[best_alpha], "The train loss is:",log_loss(Ytr
ain, predicted_train, labels=sig_clf.classes_, eps=1e-15))
print(' For alpha = ', parameter[best_alpha], "The test loss is:",log_loss(Ytes
t, predicted_test, labels=sig_clf.classes_, eps=1e-15))
```

```
 For alpha =  1000 The train loss is: 0.5816603136475661
 For alpha =  1000 The test loss is: 0.5820093471693568
```

In [0]:
```
pred_train = predict_with_best_t(predicted_train, log_error_train[best_alpha])
pred_test = predict_with_best_t(predicted_test, log_error_test[best_alpha])
confussion_matrix(pred_train,Ytrain.tolist(), 'train' )
confussion_matrix(pred_test,Ytest.tolist(), 'test' )
```







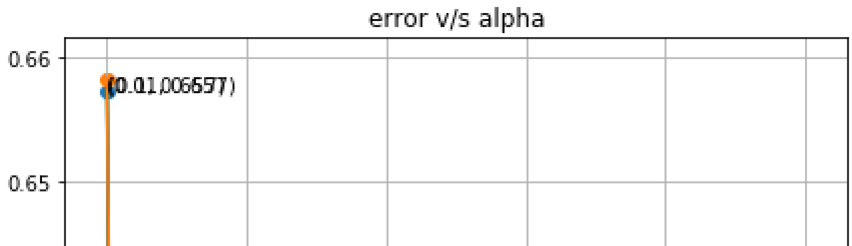
### Summary

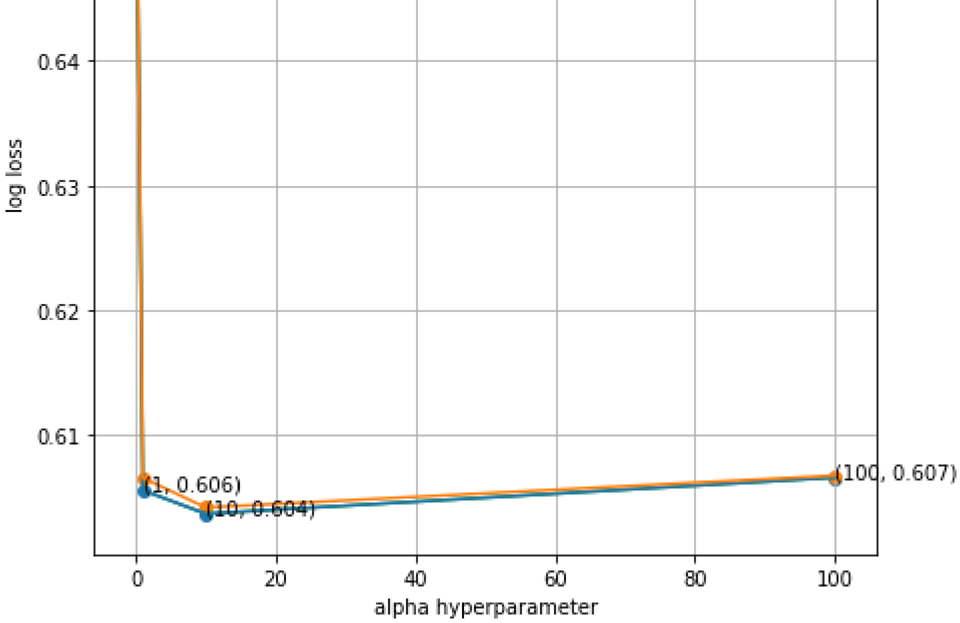
The log loss using logistic regression is 0.5816 which is better than random model but is greater than .5 which means model performance is not good with logistic regression

## Linear Svm

```
In [0]:
%%time
parameter = [10**x for x in range(-2,3)]
log_error_test = []
log_error_train = []

for i in (parameter):
    LR = SGDClassifier(alpha=i, penalty='l2',los='hinge', random_state=42) # us
ing SGDClassifier for logistic regression with loss= log
    sig_clf = CalibratedClassifierCV(LR, method="sigmoid")
    sig_clf.fit(train, Ytrain)
    predicted_train = sig_clf.predict_proba(train)
    predicted_test = sig_clf.predict_proba(test)
    log_error_test.append(log_loss(Ytest, predicted_test, labels=sig_clf.classe
s_, eps=1e-15))
    log_error_train.append(log_loss(Ytrain, predicted_train, labels=sig_clf.cla
sses_, eps=1e-15))
    print(' For values of alpha = ', i, "The log loss is:",log_loss(Ytest, pred
icted_test, labels=sig_clf.classes_, eps=1e-15))

plt.figure(figsize = (7, 7), facecolor = None)
plt.plot(parameter, log_error_test,c='g')
for i, txt in enumerate(np.round(log_error_test,3)):
    plt.annotate((parameter[i],np.round(txt,3)), (parameter[i],log_error_test
[i]))

plt.plot(parameter, log_error_test, label='Test log loss')
plt.scatter(parameter, log_error_test, label='Test los loss')
plt.plot(parameter, log_error_train, label='Train log loss')
plt.scatter(parameter, log_error_train, label='Train los loss')

plt.title('Hyper parameter v/s log loss')
plt.xlabel('alpha hyperparameter')
plt.ylabel('log loss')
plt.title('error v/s alpha')
plt.grid()

plt.show()

plt.close()
```

```
 For values of alpha =  0.01 The log loss is: 0.6573141219912
 For values of alpha =  0.1 The log loss is: 0.6573141219912
 For values of alpha =  1 The log loss is: 0.6055505220105439
 For values of alpha =  10 The log loss is: 0.603665866397361
 For values of alpha =  100 The log loss is: 0.6065704367746616
```





```
CPU times: user 42.9 s, sys: 2.99 s, total: 45.8 s
Wall time: 42.4 s
```

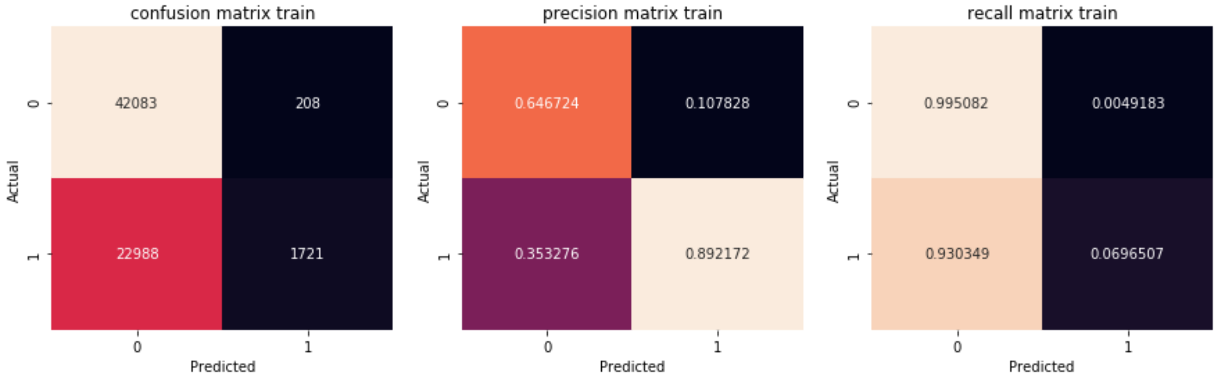
In [0]:

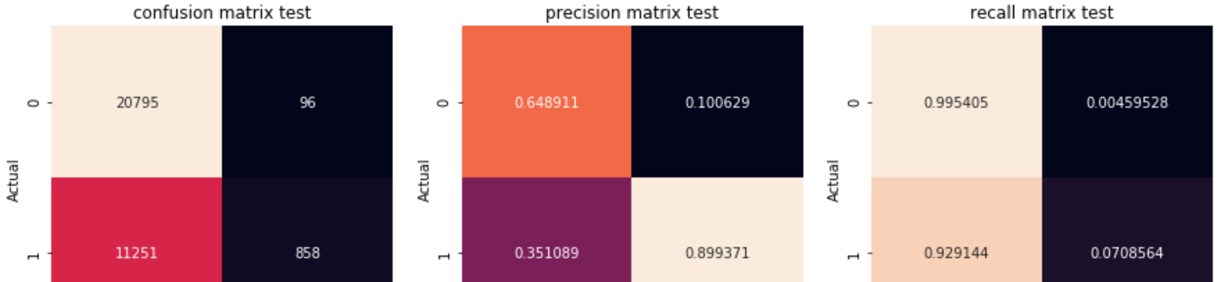
```
best_alpha = np.argmin(log_error_test)
LR = SGDClassifier(alpha=parameter[best_alpha], penalty='l2', loss='hinge', random_state=42) # using SGDClassifier for logistic regression with loss= log
sig_clf = CalibratedClassifierCV(LR, method="sigmoid")
sig_clf.fit(train, Ytrain)
predicted_train = sig_clf.predict_proba(train)
predicted_train = [i[1] for i in predicted_train ]
predicted_test = sig_clf.predict_proba(test)
predicted_test = [i[1] for i in predicted_test]

print(' For alpha = ', best_alpha, "The train loss is:",log_loss(Ytrain, predicted_train, labels=sig_clf.classes_, eps=1e-15))
print(' For alpha = ', best_alpha, "The test loss is:",log_loss(Ytest, predicted_test, labels=sig_clf.classes_, eps=1e-15))
```

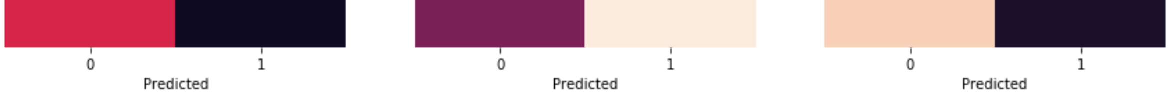
```
 For alpha =  3 The train loss is: 0.6041668550223832
 For alpha =  3 The test loss is: 0.603665866397361
```

In [0]:

```
pred_train = predict_with_best_t(predicted_train, log_error_train[best_alpha])
pred_test = predict_with_best_t(predicted_test, log_error_test[best_alpha])
confussion_matrix(pred_train,Ytrain.tolist(), 'train' )
confussion_matrix(pred_test,Ytest.tolist(), 'test' )
```







**Summary**

The log loss using SGDClassifier with hinge loss is 0.6041 which is poor than logistic regression model, which means model performance is not good with Linea Svm

# GBDT using XGboost

In [0]:

```
def plot_heatmaps(parameter, param_name, train_error, cv_error):
    # creating temperory data frame to plot heatmaps
    tmp11 = pd.DataFrame()
    tmp11[param_name[0]]= parameter[0] # k11
    tmp11[param_name[1]]=parameter[1] #K21
    tmp11['train']= train_error# train_error
    tmp11['cv'] = cv_error #cv_error

    #https://github.com/mGalarnyk/Python_Tutorials/blob/master/Request/Heat%20M
aps%20using%20Matplotlib%20and%20Seaborn.ipynb
    fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (20,20));
    pivot_table = tmp11.pivot(param_name[0], param_name[1], 'train')
    pivot_table2=tmp11.pivot(param_name[0], param_name[1], 'cv')
    axes[0].set_title('Log-loss for Train data',size=15)
    axes[0].set_xlabel(param_name[0],fontsize=15)
    axes[0].set_ylabel(param_name[1],fontsize=15)
    axes[1].set_xlabel(param_name[0],fontsize=15)
    axes[1].set_ylabel(param_name[1],fontsize=15)
    axes[1].set_title('Log-loss for cv data',size=15)
    ax = sns.heatmap(pivot_table, annot=True,annot_kws={"size": 15}, fmt='.3f',
linewidths=.5, square = True, cmap = 'cool', cbar=False,ax=axes[0])
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
    ax = sns.heatmap(pivot_table2, annot=True,annot_kws={"size": 15},fmt='.3f',
linewidths=.5, square = True, cmap = 'cool',cbar=False,ax=axes[1])
    bottom, top = ax.get_ylim()
    ax.set_ylim(bottom + 0.5, top - 0.5)
```

In [0]:

```
# to train the XGBClassifier model below link has been referred
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-
xgboost-with-codes-python/
# choosing 5 parameters to train the model

max_depth = [x for x in range(1,15, 3)]
n_estimators = [x for x in range(10,100, 10)]
learning_rate = list(np.linspace(0.01, 1, 5))
min_child_weight = list(np.linspace(0, 3, 5))
gamma = list(np.linspace(0,1,5))
```

In [207]:

```
%%time
# tunning max_depth and n_estimators
parameter1 = {'max_depth': max_depth, 'n_estimators':n_estimators}

clf = XGBClassifier(booster = 'gbtree', nthread=4) # using XGBClassifier from x
gboost for
model = RandomizedSearchCV(clf, parameter1, n_iter=10, n_jobs=1,cv=5,scoring='n
eg_log_loss', return_train_score=True)
model.fit(train_2, Ytrain)
```

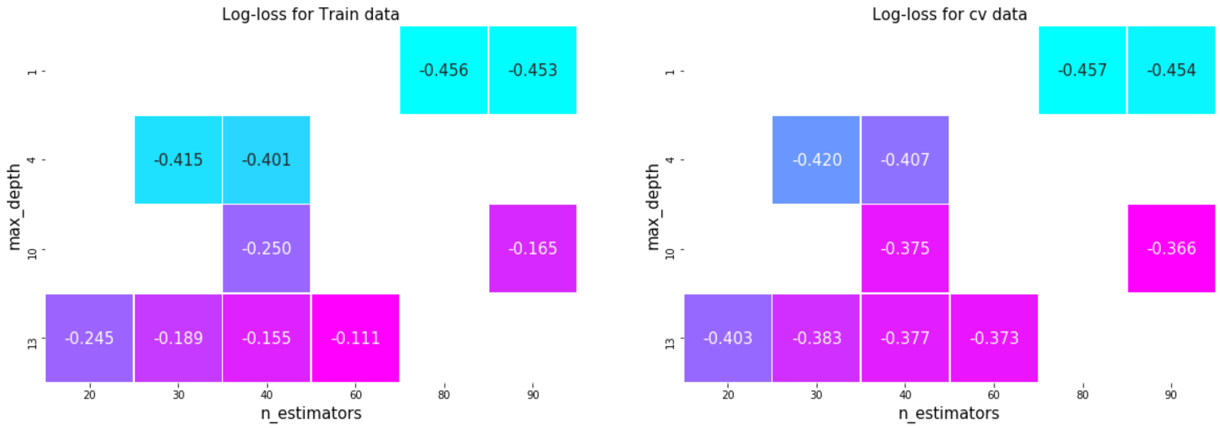
```
CPU times: user 2h 50min 21s, sys: 35.5 s, total: 2h 50min 56s
```

```
Wall time: 1h 27min 37s
```

In [209]:

```
results1 = pd.DataFrame.from_dict(model.cv_results_)
# getting important metrics from results
train_error= results1['mean_train_score']
cv_error = results1['mean_test_score']
parameter = [results1['param_max_depth'], results1['param_n_estimators']]
param_name = ['max_depth', 'n_estimators']
print("number of data points = {} \n".format(len(Ytest)))

plot_heatmaps( parameter, param_name, train_error, cv_error)
```

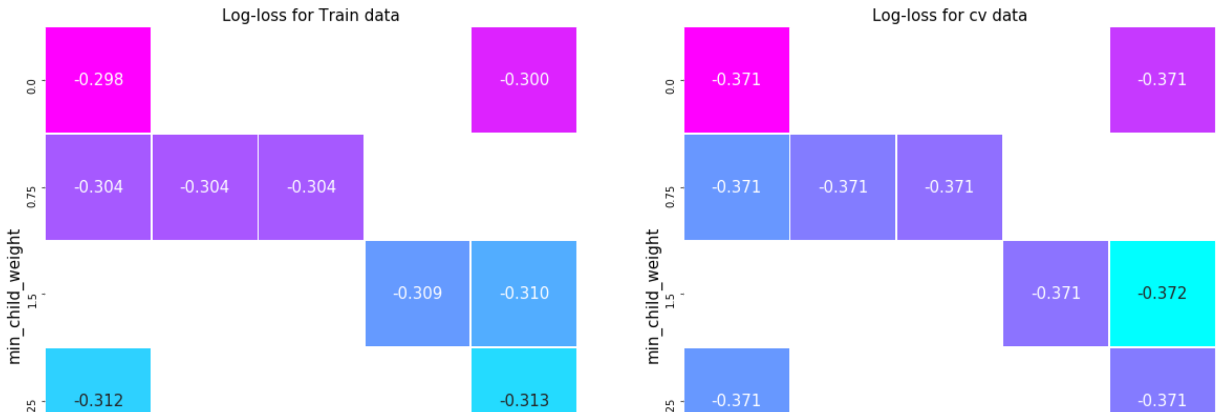
```
number of data points = 33000
```

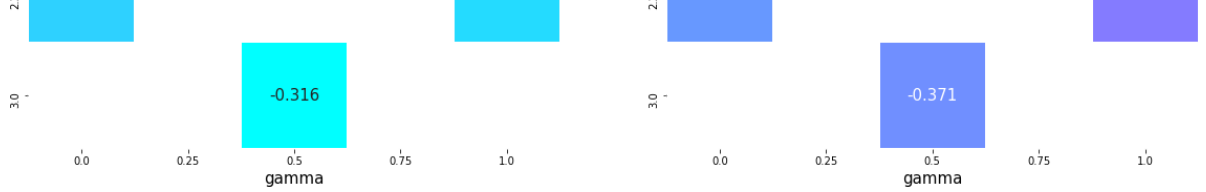


In [210]:

```
%%time
# based on the above heat maps choosing max_depth=13 and
# tuning the min_child_weight and gamma hyper parameters
parameter2 = {'min_child_weight':min_child_weight, 'gamma':gamma}

clf = XGBClassifier(max_depth=10,n_estimators =40 ,booster = 'gbtree', nthread=
4) # using XGBClassifier from xgboost for
model = RandomizedSearchCV(clf, parameter2, n_iter=10, n_jobs=4,cv=5,scoring='n
eg_log_loss', return_train_score=True)
model.fit(train, Ytrain)

results2 = pd.DataFrame.from_dict(model.cv_results_)
# getting important metrics from results
train_error= results2['mean_train_score']
cv_error = results2['mean_test_score']
parameter = [results2['param_min_child_weight'], results2['param_gamma']]
param_name = ['min_child_weight', 'gamma']
print("number of data points = {} \n".format(len(Ytest)))

plot_heatmaps( parameter, param_name, train_error, cv_error)
```

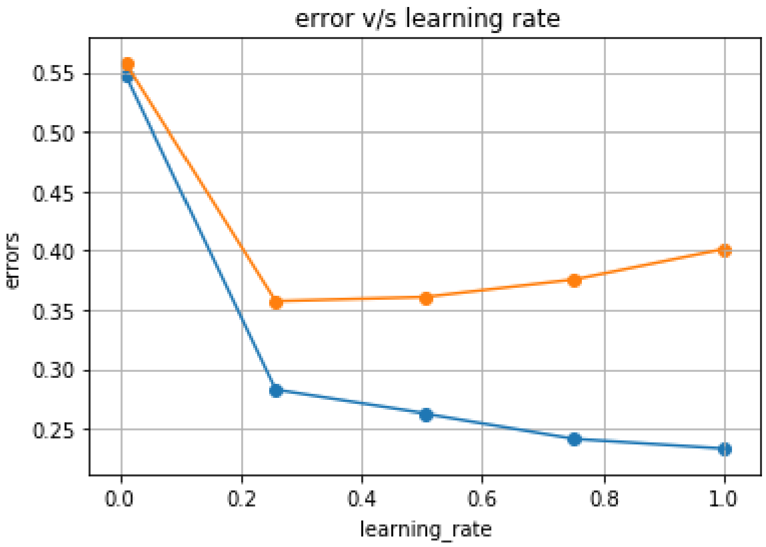
```
number of data points = 33000

CPU times: user 1min 30s, sys: 1.41 s, total: 1min 32s
Wall time: 26min 37s
```





In [213]:

```
%%time
# finding the suitable learning rate for the model
log_error_test=[]
log_error_train=[]

for i in learning_rate:
    clf = XGBClassifier(max_depth=10,min_child_weight=1.5,gamma=0.5,n_estimator
s=40,booster = 'gbtree',learning_rate= i,nthread=4) # using XGBClassifier from
 xgboost for
    #model = RandomizedSearchCV(clf, parameter3, n_iter=10, n_jobs=4,cv=5,scori
ng='neg_log_loss', return_train_score=True)
    clf.fit(train, Ytrain)
    pred_train = clf.predict_proba(train)
    pred_test = clf.predict_proba(test)
    log_error_test.append(log_loss(Ytest, pred_test, eps=1e-15, labels=clf.clas
ses_))
    log_error_train.append(log_loss(Ytrain, pred_train, eps=1e-15, labels=clf.c
lasses_))

#results = pd.DataFrame.from_dict(model.cv_results_)
# getting important metrics from results
train_error= log_error_train
test_error =  log_error_test


plt.plot(learning_rate, train_error, label="train eror")
plt.plot(learning_rate, test_error, label='test error')

plt.scatter(learning_rate, train_error)
plt.scatter(learning_rate, test_error)

plt.xlabel('learning_rate')
plt.ylabel('errors')
plt.grid()

plt.title(' error v/s learning rate')
plt.show()
plt.close()
```



```
CPU times: user 4min 6s, sys: 1.23 s, total: 4min 8s
Wall time: 2min 17s
```

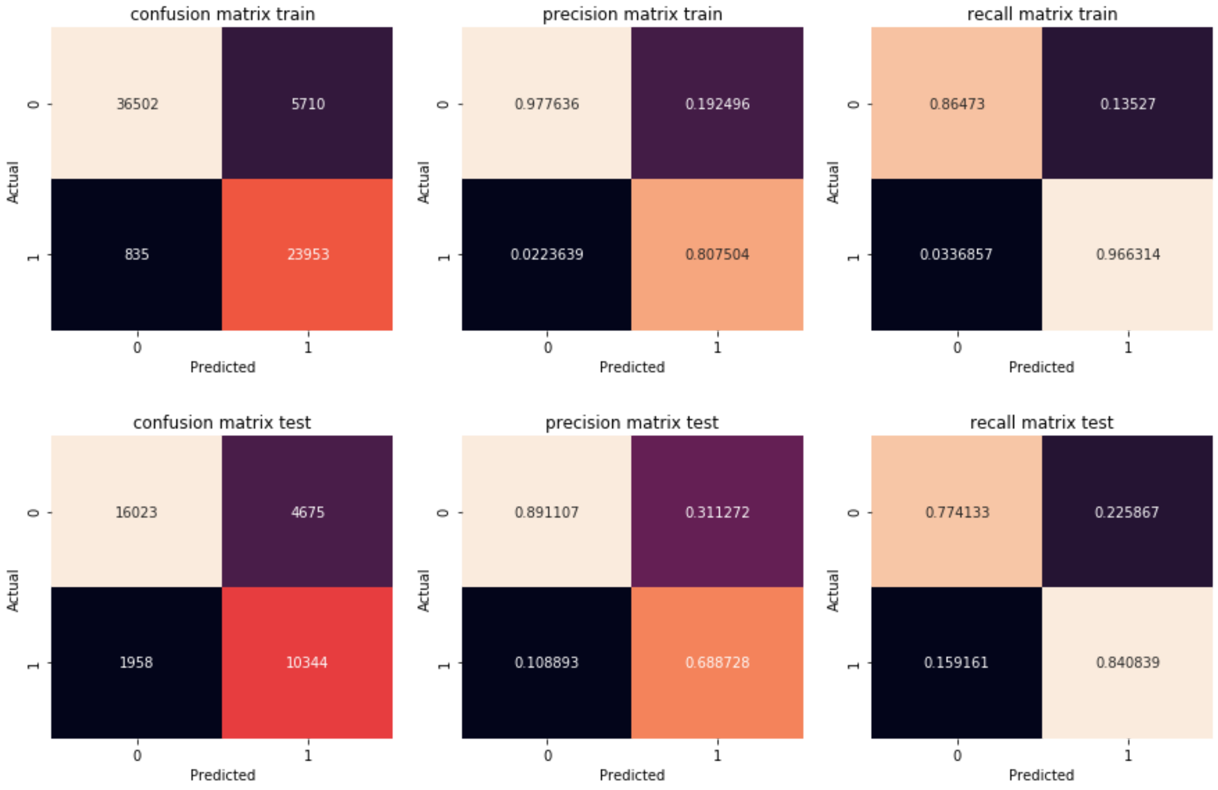
In [238]:

```
%%time

# after the important hyperparameters, training the final model
clf = XGBClassifier(max_depth=8,
                    min_child_weight=1.5,
                    gamma=0.5,
                    n_estimators=45,
                    booster = 'gbtree',
                    nthread=4,
                    colsample_bytree=1,
                    colsample_bylevel=1,
                    learning_rate=0.25,
                    verbosity=1) # using XGBClassifier from xgboost for GBDT
clf.fit(train_2, Ytrain, verbose=1)
log_error_test=[]

predicted_train = clf.predict_proba(train_2)
predicted_train = [i[1] for i in predicted_train ]
predicted_test = clf.predict_proba(test_2)
predicted_test = [i[1] for i in predicted_test ]
log_error_test.append(log_loss(Ytest, predicted_test, labels=clf.classes_, eps=
1e-15))
print( "The log loss train is:",log_loss(Ytrain, predicted_train, labels=clf.cl
asses_, eps=1e-15))
print( "The log loss test is:",log_loss(Ytest, predicted_test, labels=clf.class
es_, eps=1e-15))
```

```
The log loss train is: 0.2353068252385698
The log loss test is: 0.36935211935219076
CPU times: user 3min 37s, sys: 509 ms, total: 3min 38s
Wall time: 1min 51s
```

In [239]:

```
%%time
log_error=  log_error_test[0] # giving the value of log loss
pred_train = predict_with_best_t(predicted_train, log_error)
pred_test = predict_with_best_t(predicted_test, log_error)
confussion_matrix(pred_train,Ytrain.tolist(), 'train' )
confussion_matrix(pred_test,Ytest.tolist(), 'test' )
```



```
CPU times: user 1.04 s, sys: 204 ms, total: 1.24 s
Wall time: 1.02 s
```

Summary

We can see that log loss is 0.3693 which is much better than logistic regression model and linear svm model, also log loss is lesser than 0.5 also the precision and recall are holding well for this model

In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
#http://zetcode.com/python/prettytable
x = PrettyTable()
x.field_names = ["model","Text Vetcor", "Train Loss", "Test Loss"]
x.add_row(["Logistic Regression","TFIDF",  0.5816, 0.5820])
x.add_row(["Linear Svm", "TFIDF", 0.6041, 0.6036])
x.add_row(["GBDT", "TFIDF-W2Vec" ,0.2353, 0.3693])
```

# Conclusion:

In [246]:

```
print(x)
```

```
+---------------------+-------------+------------+-----------+
|        model        | Text Vetcor | Train Loss | Test Loss |
+---------------------+-------------+------------+-----------+
| Logistic Regression |    TFIDF    |   0.5816   |   0.582   |
|      Linear Svm     |    TFIDF    |   0.6041   |   0.6036  |
|         GBDT        | TFIDF-W2Vec |   0.2353   |   0.3693  |
+---------------------+-------------+------------+-----------+
```

Based on the above table we can conclude that GBDT model with weighted w2vec performs the best among logistic regression and linear svm. although the the difference b/w train loss and test loss is considerable it is giving good precisoin and recall.