FINAL_QUORA_QUESTION_PAIR (1)

May 31, 2019

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 ____ Useful Links ____
- Discussions : 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/AACZd
- Blog 1 : 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

1.3 Real world/Business Objectives and Constraints

attachment:Quora-1.png

Quora-1.png

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

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

Metric(s): * log-loss : 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 [0]: 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
        from bs4 import BeautifulSoup
```

3.1 Reading data and basic stats

```
In [0]: df = pd.read_csv("train.csv")

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

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

```
In [0]: df.head()
```

```
Out[0]:    id  qid1  qid2                                                question1  \
        0   0     1     2  What is the step by step guide to invest in sh...
        1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...
        2   2     5     6  How can I increase the speed of my internet co...
        3   3     7     8  Why am I mentally very lonely? How can I solve...
        4   4     9    10  Which one dissolve in water quikly sugar, salt...

                                                   question2  is_duplicate
        0  What is the step by step guide to invest in sh...             0
        1  What would happen if the Indian government sto...             0
        2  How can Internet speed be increased by hacking...             0
        3  Find the remainder when [math]23^{24}[/math] i...             0
        4              Which fish would survive in salt water?             0
```

```
In [0]: 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:

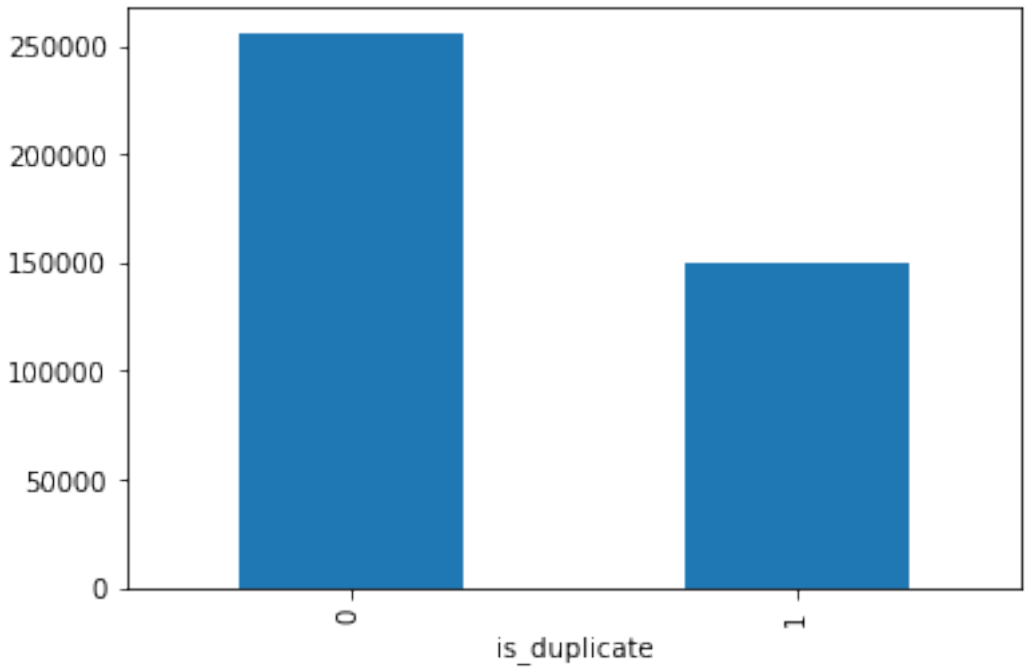
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

```
In [0]: df.groupby("is_duplicate")['id'].count().plot.bar()
```

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




```
In [0]: print('~> Total number of question pairs for training:\n   {}'.format(len(df)))
```

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

```
In [0]: print('~> Question pairs are not Similar (is_duplicate = 0):\n   {}%'.format(100 - round
        print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_
```

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

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

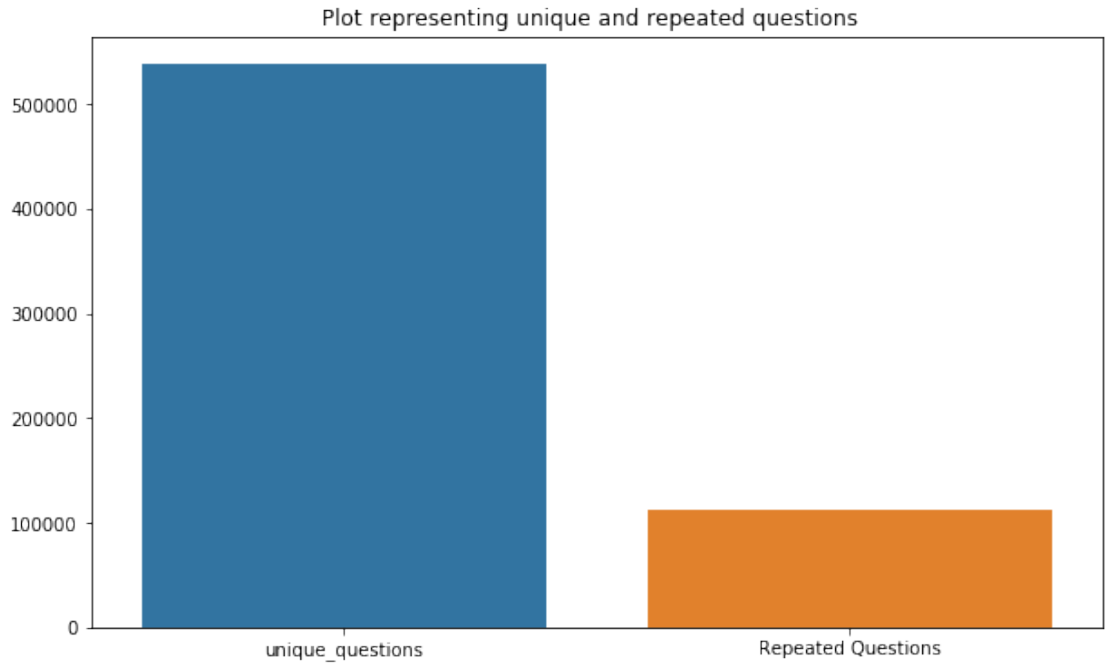
3.2.2 Number of unique questions

```
In [0]: 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'.format(qs

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

        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 (20.77953945937505%)

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

```
In [0]: 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

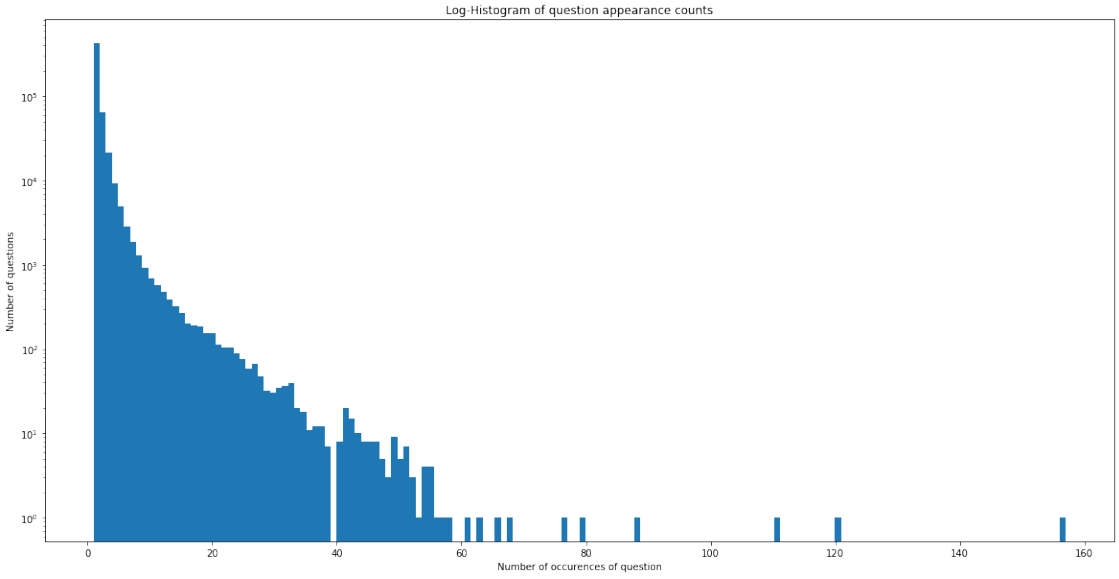
```
In [0]: #checking whether there are any repeated pair of questions

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

        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 [0]: 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(max(qids.val
```

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



3.2.5 Checking for NULL values

```
In [0]: #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 [0]: # 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 [0]: 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[0]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	freq_q1-q2
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	14	12	10.0	23.0	0.434783	2	0
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	8	13	4.0	20.0	0.200000	5	3
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	14	10	4.0	24.0	0.166667	2	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	1	1	50	65	11	9	0.0	19.0	0.000000	2	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39	13	7	2.0	20.0	0.100000	4	2

3.3.1 Analysis of some of the extracted features

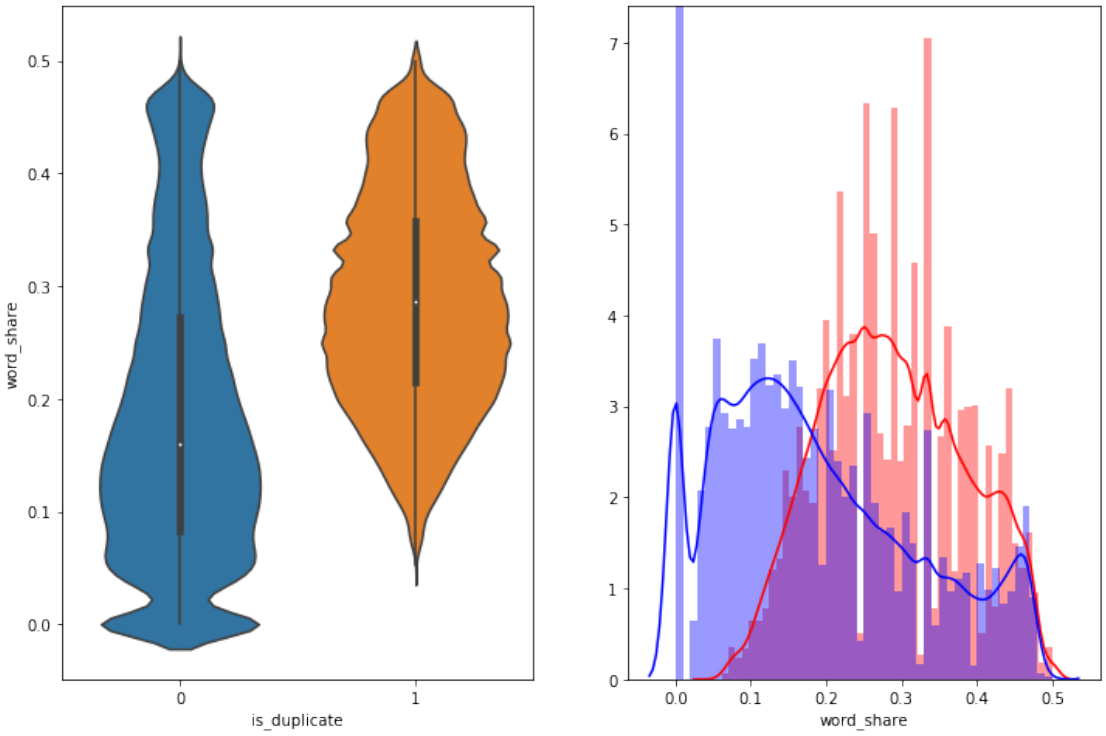
- Here are some questions have only one single words.

```
In [0]: 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_words']== 1]
        print ("Number of Questions with minimum length [question2] :", df[df['q2_n_words']== 1]
```

```
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

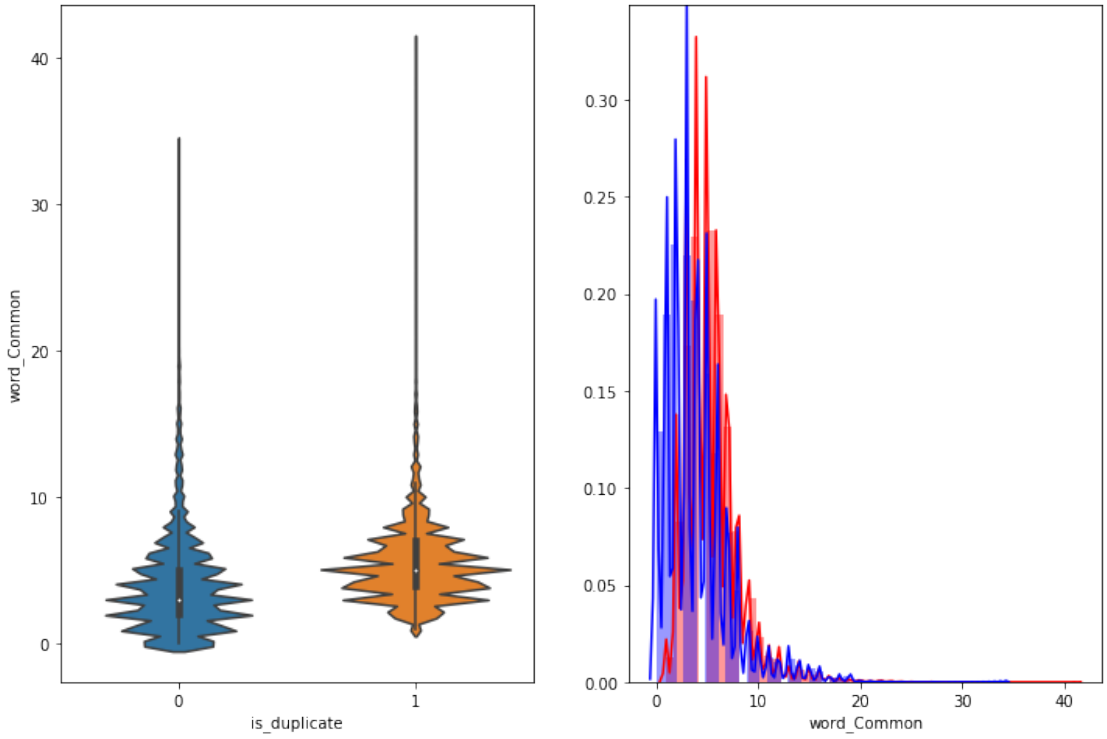
```
In [0]: plt.figure(figsize=(12, 8))

        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", color = 'red
        sns.distplot(df[df['is_duplicate'] == 0.0]['word_share'][0:] , label = "0" , color = 'bl
        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 [0]: 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", color = 're
        sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:] , label = "0" , color = 'b
        plt.show()
```



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

0.0.1 1.2.1 : EDA: Advanced Feature Extraction.

```
In [0]: #https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-decode-
        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 previous note
```

```
In [0]: df.head(2)
```

Out[0]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	freq_q1-q2
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	14	12	10.0	23.0	0.434783	2	0
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	8	13	4.0	20.0	0.200000	5	3

3.4 Preprocessing of Text

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

```
In [0]: # 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("′", "'").replace("’", "
                                    .replace("won't", "will not").replace("cannot", "can not").re
                                    .replace("n't", " not").replace("what's", "what is").replace(
                                    .replace("'ve", " have").replace("i'm", "i am").replace("'re"
```

```
                        .replace("he's", "he is").replace("she's", "she is").replace(
                        .replace("%", " percent ").replace("[U+20B9]", " rupee ").rep
                        .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 Q2csc_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 Q2ctc_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 Q2ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))
- **last_word_eq** : Check if First word of both questions is equal or notlast_word_eq = int(q1_tokens[-1] == q2_tokens[-1])
- **first_word_eq** : Check if First word of both questions is equal or notfirst_word_eq = int(q1_tokens[0] == q2_tokens[0])
- **abs_len_diff** : Abs. length differenceabs_len_diff = abs(len(q1_tokens) - len(q2_tokens))
- **mean_len** : Average Token Length of both Questionsmean_len = (len(q1_tokens) + len(q2_tokens))/2

- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **fuzz_partial_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/
- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage 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 Q2longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

```
In [0]: 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)) + SAFE_DI
            token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DI
            token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DI
            token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DI
            token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE
            token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE
```

```
    # 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["question2"

    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-matching-
    # https://stackoverflow.com/questions/31806695/when-to-use-which-fuzz-function-to-co
```

```
        # https://github.com/seatgeek/fuzzywuzzy
        print("fuzzy features..")

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

In [0]:
```
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[0]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	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
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	...	0.785709	0.0	1.0	2.0	13.0	100	93	93	100	0.982759
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	...	0.466664	0.0	1.0	5.0	12.5	86	63	66	75	0.596154

[2 rows x 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 [0]: 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}} to {1,2,3,4
        p = np.dstack([df_duplicate["question1"], df_duplicate["question2"]]).flatten()
        n = np.dstack([dfp_nonduplicate["question1"], dfp_nonduplicate["question2"]]).flatten()

        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 [0]: # 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 __

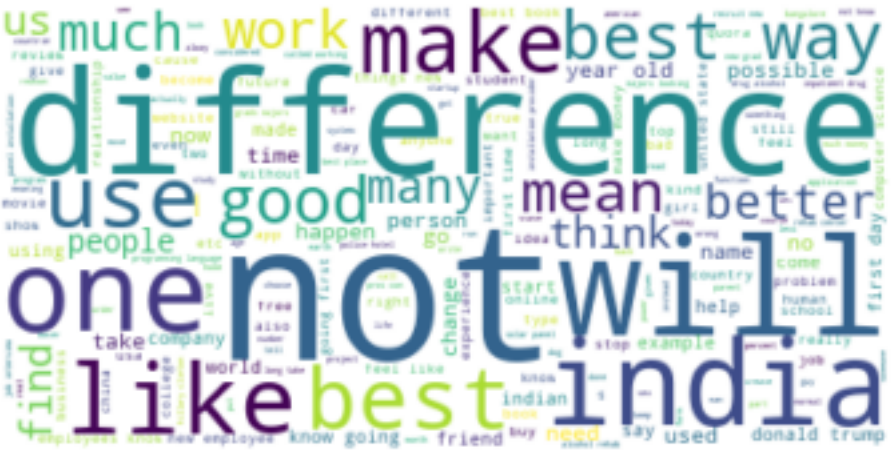
```
In [0]: wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopwords)
        wc.generate(textp_w)
        print ("Word Cloud for Duplicate Question pairs")
        plt.imshow(wc, interpolation='bilinear')
        plt.axis("off")
        plt.show()
```

Word Cloud for Duplicate Question pairs

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

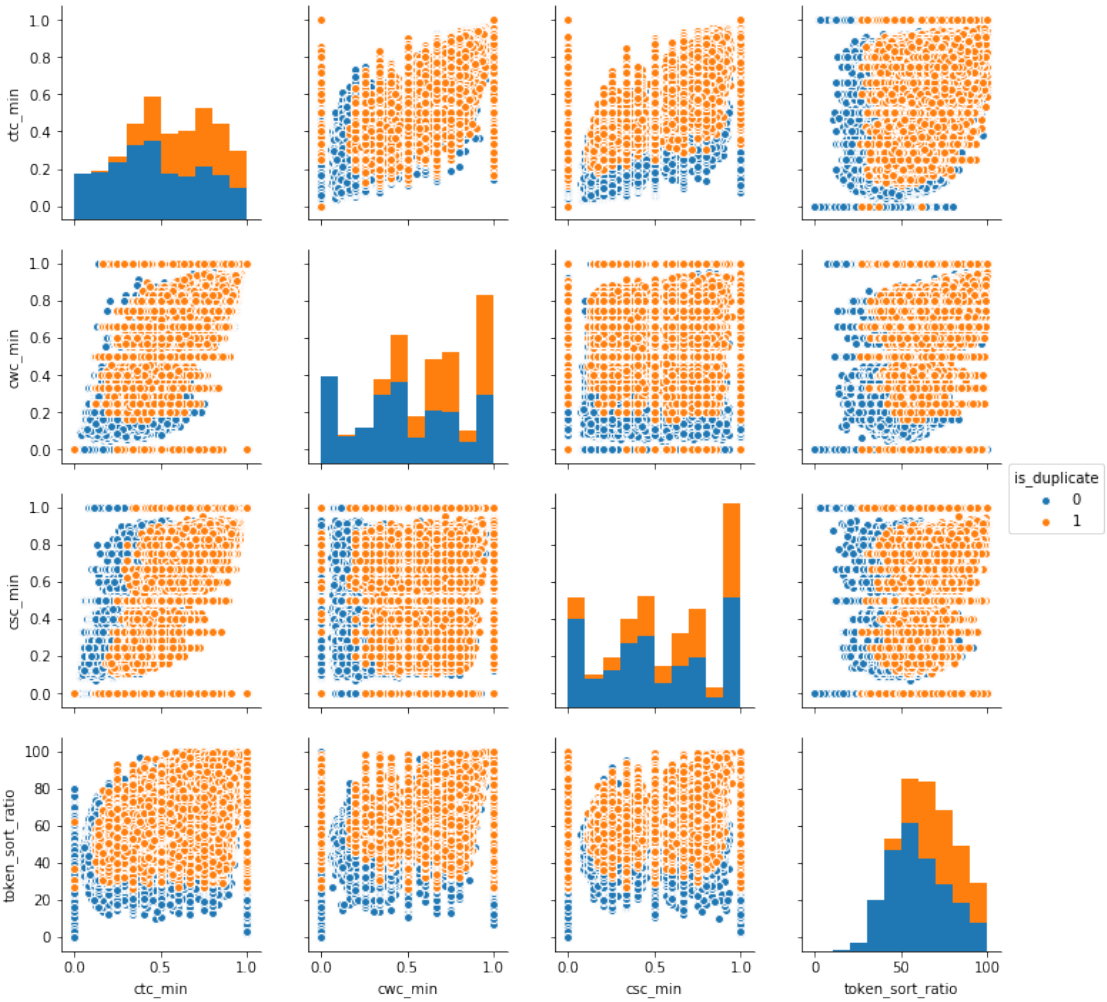
```
In [0]: wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
        # generate word cloud
        wc.generate(textn_w)
        print ("Word Cloud for non-Duplicate Question pairs:")
        plt.imshow(wc, interpolation='bilinear')
        plt.axis("off")
        plt.show()
```

Word Cloud for non-Duplicate Question pairs:



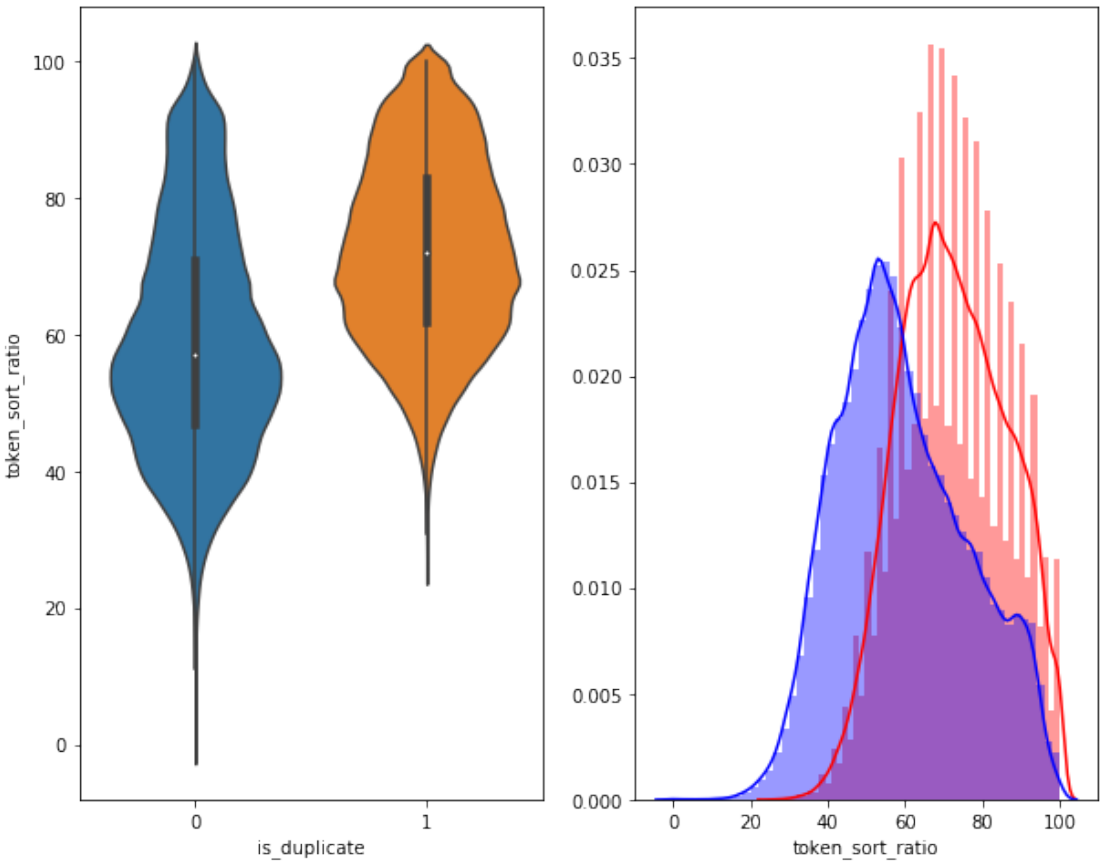
3.5.1.2 Pair plot of features ['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio']

```
In [0]: n = df.shape[0]
        sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']][0
        plt.show()
```

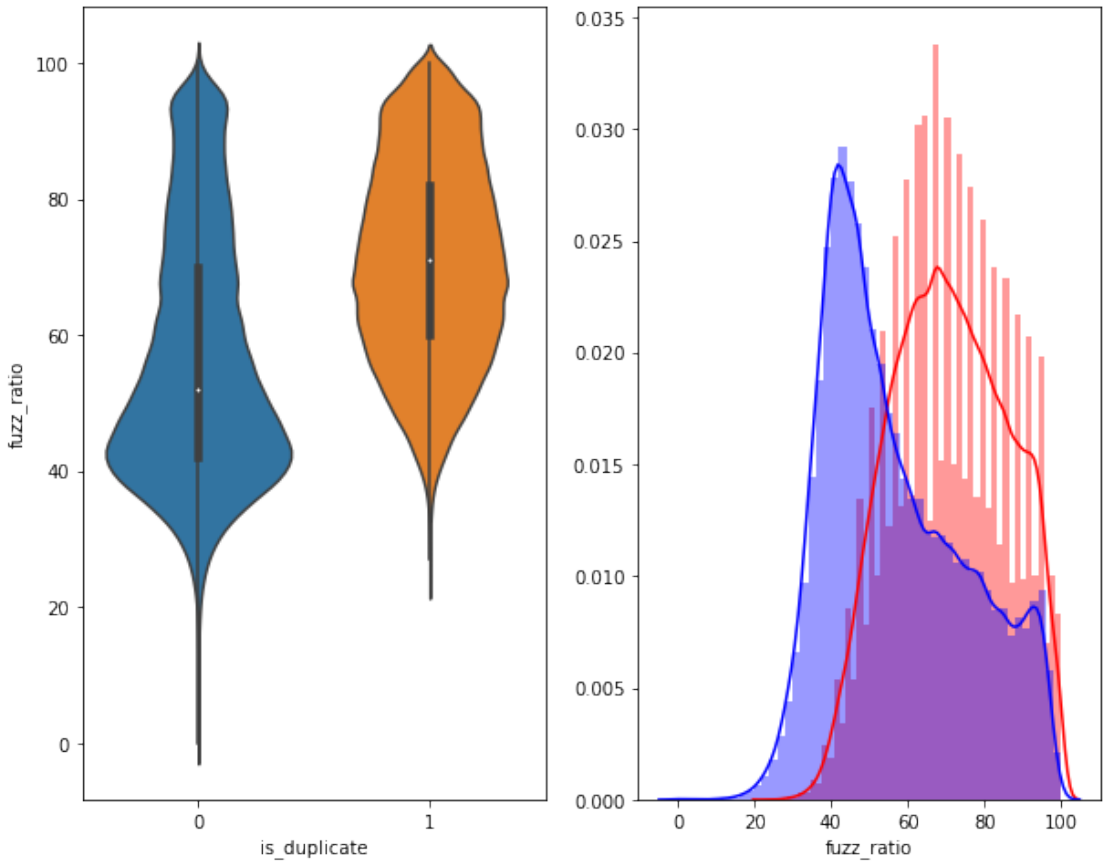



```
In [0]: # Distribution of the token_sort_ratio
        plt.figure(figsize=(10, 8))

        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 = "1", color
```

```
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" , color
plt.show()
```



In [0]:
```
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", color = 'red
sns.distplot(df[df['is_duplicate'] == 0.0]['fuzz_ratio'][0:] , label = "0" , color = 'bl
plt.show()
```



3.5.2 Visualization

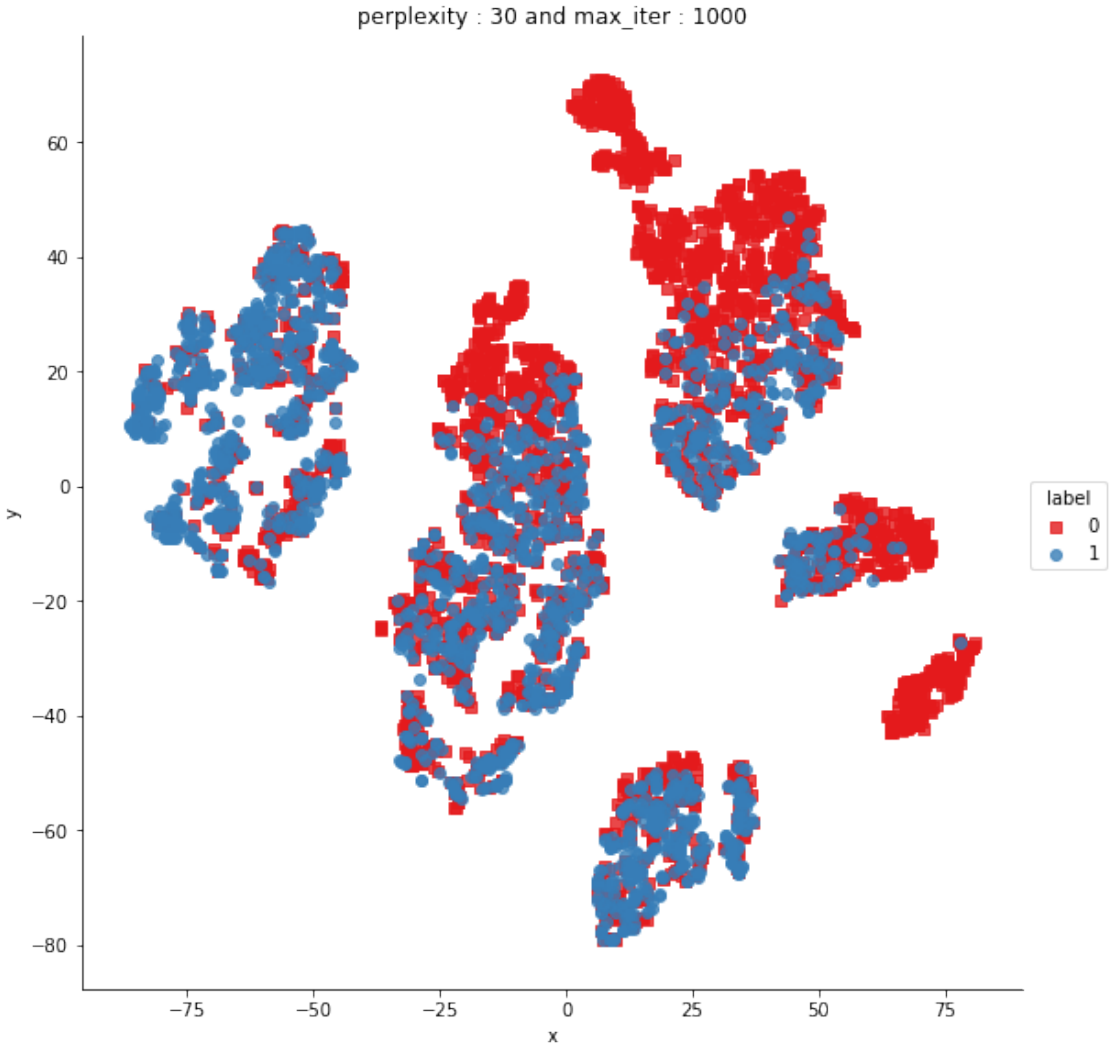
```
In [0]: # Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the d

        from sklearn.preprocessing import MinMaxScaler

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

```
In [0]: 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 iterations in 13.099s)
[t-SNE] Iteration 100: error = 70.3858795, gradient norm = 0.0100968 (50 iterations in 9.067s)
[t-SNE] Iteration 150: error = 68.6138382, gradient norm = 0.0058392 (50 iterations in 9.602s)
[t-SNE] Iteration 200: error = 67.7700119, gradient norm = 0.0036596 (50 iterations in 9.121s)
[t-SNE] Iteration 250: error = 67.2725067, gradient norm = 0.0034962 (50 iterations 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 iterations in 8.289s)
[t-SNE] Iteration 350: error = 1.3720417, gradient norm = 0.0004822 (50 iterations in 10.526s)
[t-SNE] Iteration 400: error = 1.2039998, gradient norm = 0.0002768 (50 iterations in 9.600s)
[t-SNE] Iteration 450: error = 1.1133438, gradient norm = 0.0001881 (50 iterations in 11.827s)
[t-SNE] Iteration 500: error = 1.0579143, gradient norm = 0.0001434 (50 iterations in 8.941s)
[t-SNE] Iteration 550: error = 1.0221983, gradient norm = 0.0001164 (50 iterations in 11.092s)
[t-SNE] Iteration 600: error = 0.9987167, gradient norm = 0.0001039 (50 iterations in 11.467s)
[t-SNE] Iteration 650: error = 0.9831534, gradient norm = 0.0000938 (50 iterations in 11.799s)
[t-SNE] Iteration 700: error = 0.9722011, gradient norm = 0.0000858 (50 iterations in 12.028s)
[t-SNE] Iteration 750: error = 0.9643636, gradient norm = 0.0000799 (50 iterations in 12.120s)
[t-SNE] Iteration 800: error = 0.9584482, gradient norm = 0.0000785 (50 iterations in 11.867s)
[t-SNE] Iteration 850: error = 0.9538348, gradient norm = 0.0000739 (50 iterations in 11.461s)
[t-SNE] Iteration 900: error = 0.9496906, gradient norm = 0.0000712 (50 iterations in 11.023s)
[t-SNE] Iteration 950: error = 0.9463405, gradient norm = 0.0000673 (50 iterations in 11.755s)
[t-SNE] Iteration 1000: error = 0.9432716, gradient norm = 0.0000662 (50 iterations in 11.493s)
[t-SNE] Error after 1000 iterations: 0.943272
```

In [0]:
```
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="Set1",mark
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```




```
In [0]: 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 iterations in 34.931s)
[t-SNE] Iteration 100: error = 69.2682266, gradient norm = 0.0248657 (50 iterations in 15.147s)
[t-SNE] Iteration 150: error = 67.7877655, gradient norm = 0.0150941 (50 iterations in 13.761s)
[t-SNE] Iteration 200: error = 67.1991119, gradient norm = 0.0126559 (50 iterations in 13.425s)
[t-SNE] Iteration 250: error = 66.8560715, gradient norm = 0.0074975 (50 iterations 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 iterations in 13.302s)
[t-SNE] Iteration 350: error = 0.9948602, gradient norm = 0.0001997 (50 iterations in 18.898s)
[t-SNE] Iteration 400: error = 0.9168936, gradient norm = 0.0001430 (50 iterations in 13.397s)
[t-SNE] Iteration 450: error = 0.8863022, gradient norm = 0.0000975 (50 iterations in 16.379s)
[t-SNE] Iteration 500: error = 0.8681002, gradient norm = 0.0000854 (50 iterations in 17.791s)
[t-SNE] Iteration 550: error = 0.8564141, gradient norm = 0.0000694 (50 iterations in 17.060s)
[t-SNE] Iteration 600: error = 0.8470711, gradient norm = 0.0000640 (50 iterations in 15.454s)
[t-SNE] Iteration 650: error = 0.8389117, gradient norm = 0.0000561 (50 iterations in 17.562s)
[t-SNE] Iteration 700: error = 0.8325295, gradient norm = 0.0000529 (50 iterations in 13.443s)
[t-SNE] Iteration 750: error = 0.8268463, gradient norm = 0.0000528 (50 iterations in 17.981s)
[t-SNE] Iteration 800: error = 0.8219477, gradient norm = 0.0000477 (50 iterations in 17.448s)
[t-SNE] Iteration 850: error = 0.8180174, gradient norm = 0.0000490 (50 iterations in 18.376s)
[t-SNE] Iteration 900: error = 0.8150476, gradient norm = 0.0000456 (50 iterations in 17.778s)
[t-SNE] Iteration 950: error = 0.8122067, gradient norm = 0.0000472 (50 iterations in 16.983s)
[t-SNE] Iteration 1000: error = 0.8095787, gradient norm = 0.0000489 (50 iterations in 18.581s)
[t-SNE] Error after 1000 iterations: 0.809579
```

```
In [0]: import pandas as pd
        import matplotlib.pyplot as plt
        import re
        import time
        import warnings
        import numpy as np
        from nltk.corpus import stopwords
        from sklearn.preprocessing import normalize
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer
        warnings.filterwarnings("ignore")
        import sys
        import os
        import pandas as pd
        import numpy as np
        from tqdm import tqdm

        # exctract word2vec vectors
```

```
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
# from sklearn.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
```

3.6 Featurizing text data with tfidf weighted word-vectors

```
In [ ]: # Install the PyDrive wrapper & import libraries.
        # This only needs to be done once per notebook.
        !pip install -U -q PyDrive
        from pydrive.auth import GoogleAuth
        from pydrive.drive import GoogleDrive
        from google.colab import auth
        from oauth2client.client import GoogleCredentials

        # Authenticate and create the PyDrive client.
        # This only needs to be done once per notebook.
        auth.authenticate_user()
        gauth = GoogleAuth()
        gauth.credentials = GoogleCredentials.get_application_default()
        drive = GoogleDrive(gauth)

        # Download a file based on its file ID.
        #
        # A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
        listed = drive.ListFile().GetList()
        for file in listed:
            print('title {}, id {}'.format(file['title'], file['id']))
```

```
In [0]: #https://drive.google.com/file/d/10QDGTSI5PEV9e7CTpfzsXRpUwRIsJA-J/view?usp=sharing
        download = drive.CreateFile({'id': '10QDGTSI5PEV9e7CTpfzsXRpUwRIsJA-J'})
        download.GetContentFile('train.csv')
```

```
In [0]: #https://drive.google.com/file/d/10QDGTSI5PEV9e7CTpfzsXRpUwRIsJA-J/view?usp=sharing
        download = drive.CreateFile({'id': '10QDGTSI5PEV9e7CTpfzsXRpUwRIsJA-J'})
        download.GetContentFile('train.csv')
```

```
In [0]: # 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 [0]: df.head()
```

```
Out[0]:    id  qid1  ...                                          question2 is_duplicate
        0   0     1  ...  What is the step by step guide to invest in sh...            0
```

```
1   1     3  ...  What would happen if the Indian government sto...           0
2   2     5  ...  How can Internet speed be increased by hacking...           0
3   3     7  ...  Find the remainder when [math]23^{24}[/math] i...           0
4   4     9  ...             Which fish would survive in salt water?           0

[5 rows x 6 columns]
```

```
In [0]: df=df[:100000]
        df_train=df[:70000]
        df_test=df[70000:100000]
```

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.feature_extraction.text import CountVectorizer
        # merge texts
        questions_train= list(df_train['question1']) + list(df_train['question2'])
        questions_test= list(df_test['question1']) + list(df_test['question2'])


        tfidf = TfidfVectorizer(lowercase=False, )
        tfidf_train =tfidf.fit_transform(questions_train)
        tfidf_test  =tfidf.transform(questions_test)

        # 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
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

```
In [0]: # 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_train['question1'])):
            doc1 = nlp(qu1)
            # 384 is the number of dimensions of vectors
            mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
            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_train['q1_feats_tr'] = list(vecs1)
```

In [0]:
```
# 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_train['question2'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    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_train['q2_feats_tr'] = list(vecs1)
```

In [0]:
```
vecs2 = []
for qu2 in tqdm(list(df_test['question1'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
    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_test['q1_feats_ts'] = list(vecs2)
```

```
In [0]: vecs2 = []
        for qu2 in tqdm(list(df_test['question2'])):
            doc2 = nlp(qu2)
            mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
            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_test['q2_feats_ts'] = list(vecs2)

In [0]: #https://drive.google.com/file/d/1JncN1Fyt-ND_yZXOzqEfcRsYMTKqtu7Q/view?usp=sharing
        download = drive.CreateFile({'id': '1JncN1Fyt-ND_yZXOzqEfcRsYMTKqtu7Q'})
        download.GetContentFile('nlp_features_train.csv')

In [0]: #https://drive.google.com/file/d/1gTfCTD3fz-3NJnfYLm59nZFN3WC3fzfD/view?usp=sharing
        download = drive.CreateFile({'id': '1gTfCTD3fz-3NJnfYLm59nZFN3WC3fzfD'})
        download.GetContentFile('df_fe_without_preprocessing_train.csv')

In [0]: #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='latin-1')
        else:
            print("download df_fe_without_preprocessing_train.csv from drive or run previous not

In [0]: df_test.head()

In [0]: df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
        df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
        df3 = df_train.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
        df4 = df_test.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)


        df3_q1 = pd.DataFrame(df3.q1_feats_tr.values.tolist(), index= df3.index)
        df3_q2 = pd.DataFrame(df3.q2_feats_tr.values.tolist(), index= df3.index)
```

```
        df4_q1 = pd.DataFrame(df4.q1_feats_ts.values.tolist(), index= df4.index)
        df4_q2 = pd.DataFrame(df4.q2_feats_ts.values.tolist(), index= df4.index)
```

In [0]:
```
        df1_train=df1[:70000]
        df1_test=df1[70000:100000]
        df2_train=df2[:70000]
        df2_test=df2[70000:100000]
```

In [0]:
```
        # dataframe of nlp features
        df1.head()
```

Out[0]:

	id	is_duplicate	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
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	100	93	93	100	0.982759
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	86	63	66	75	0.596154
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	66	66	54	54	0.166667
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	36	36	35	40	0.039216
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	67	47	46	56	0.175000

In [0]:
```
        # data before preprocessing
        df2.head()
```

Out[0]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words
0	0	1	1	66	57	14	12
1	1	4	1	51	88	8	13
2	2	1	1	73	59	14	10
3	3	1	1	50	65	11	9
4	4	3	1	76	39	13	7

```
   word_Common  word_Total  word_share  freq_q1+q2  freq_q1-q2
0         10.0        23.0    0.434783           2           0
1          4.0        20.0    0.200000           5           3
2          4.0        24.0    0.166667           2           0
3          0.0        19.0    0.000000           2           0
4          2.0        20.0    0.100000           4           2
```

In [0]:
```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[0]:
```
           0           1           2           3          4          5    \
0  121.929927  100.083900   72.497894  115.641800 -48.370870  34.619058
1  -78.070939   54.843781   82.738482   98.191872 -51.234859  55.013510
2   -5.355015   73.671810   14.376365  104.130241   1.433537  35.229116
3    5.778359  -34.712038   48.999631   59.699204  40.661263 -41.658731
4   51.138220   38.587312  123.639488   53.333041 -47.062739  37.356212

           6           7           8          9      ...           374  \
0 -172.057787  -92.502617  113.223315  50.562441    ...     12.397642
1  -39.140730  -82.692352   45.161489  -9.556289    ...    -21.987077
2 -148.519385  -97.124595   41.972195  50.948731    ...      3.027700
3  -36.808594   24.170655    0.235600 -29.407290    ...     13.100007
4 -298.722753 -106.421119  106.248914  65.880707    ...     13.906532

          375        376        377        378        379        380       381  \
0  40.909519   8.150261 -15.170692  18.007709   6.166999 -30.124163  3.700902
1 -12.389279  20.667979   2.202714 -17.142454  -5.880972 -10.123963 -4.890663
2  14.025767  -2.960312  -3.206544   4.355141   2.936152 -20.199555  9.816351
3   1.405670  -1.891076  -7.882638  18.000561  12.106918 -10.507835  5.243834
4  43.461721  11.519207 -22.468284  45.431128   8.161224 -35.373910  7.728865

          382       383
0  -1.757693 -1.818058
1 -13.018389 -5.219310
2  11.894366 -8.798819
3  10.158340  5.886351
4   9.592849  5.447336

[5 rows x 384 columns]
```

In [0]:
```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[0]:
```
           0          1          2          3          4          5    \
0  125.983301  95.636485  42.114702  95.449980 -37.386295   39.400078
1 -106.871904  80.290331  79.066297  59.302092 -42.175328  117.616655
2    7.072875  15.513378   1.846914  85.937583 -33.808811   94.702337
3   39.421531  44.136989 -24.010929  85.265863  -0.339022   -9.323137
```

```
4   31.950101  62.854106   1.778164  36.218768 -45.130875   66.674880

           6           7           8          9      ...            374  \
0 -148.116070  -87.851475  110.371966  62.272814    ...      16.165592
1 -144.364237 -127.131513   22.962533  25.397575    ...      -4.901128
2 -122.256856 -114.009530   53.922293  60.131814    ...       8.359966
3  -60.499651  -37.044763   49.407848 -23.350150    ...       3.311411
4 -106.342341  -22.901008   59.835938  62.663961    ...      -2.403870

         375        376        377        378        379        380  \
0  33.030668   7.019996 -14.793959  15.437511   8.199658 -25.070834
1  -4.565393  41.520751  -0.727564 -16.413776  -7.373778   2.638877
2  -2.165985  10.936580 -16.531660  14.681230  15.633759  -1.210901
3   3.788879  13.398598  -6.592596   6.437365   5.993293   2.732392
4  11.991204   8.088483 -15.090201   8.375166   1.727225  -6.601129

         381        382        383
0   1.571619   1.603738   0.305645
1  -7.403457   2.703070   0.408040
2  14.183826  11.703135  10.148075
3  -3.727647   5.614115   6.023693
4  11.317413  11.544603   2.478689

[5 rows x 384 columns]
```

```
In [0]: 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]+df3_q1.shape
```

```
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 [0]: # storing the final features to csv file
        if not os.path.isfile('final_features.csv'):
            df3_q1['id']=df1_train['id']
            df3_q2['id']=df1_train['id']

            df4_q1['id']=df1_test['id']
            df4_q2['id']=df1_test['id']

            df1_train  = df1_train.merge(df2_train, on='id',how='left')
            df2_train  = df3_q1.merge(df3_q2, on='id',how='left')
```

```
    result_train  = df1_train.merge(df2_train, on='id',how='left')
    result_train.to_csv('final_train_features.csv')

    df1_test  = df1_test.merge(df2_test, on='id',how='left')
    df2_test  = df4_q1.merge(df4_q2, on='id',how='left')
    result_test  = df1_test.merge(df2_test, on='id',how='left')
    result_test.to_csv('final_test_features.csv')
```

```
In [0]: result_train = pd.read_csv("final_train_features.csv")
        result_test = pd.read_csv("final_test_features.csv")
```

```
In [0]: y_train=result_train['is_duplicate']
        y_test=result_test['is_duplicate']
```

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

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

```
In [0]: X_train=result_train
        X_test=result_test
```

```
In [0]: X_test.head()
```

```
Out[0]:     cwc_min   cwc_max   csc_min  ...      381_y      382_y     383_y
        0  0.666644  0.666644  0.999967  ...   8.319497  16.277564  1.504150
        1  0.399992  0.249997  0.000000  ...   9.271396   3.512845  5.342707
        2  0.666644  0.499988  0.666644  ...  21.379616  14.346018 -0.937722
        3  0.999967  0.749981  0.999975  ...   1.527024  17.901435 -2.211148
        4  0.249997  0.249997  0.142855  ...   2.247266  10.012977  9.257347

        [5 rows x 794 columns]
```

```
In [0]: X_test.head()
```

```
Out[0]:     cwc_min   cwc_max   csc_min  ...      381_y      382_y     383_y
        0  0.666644  0.666644  0.999967  ...   8.319497  16.277564  1.504150
        1  0.399992  0.249997  0.000000  ...   9.271396   3.512845  5.342707
        2  0.666644  0.499988  0.666644  ...  21.379616  14.346018 -0.937722
        3  0.999967  0.749981  0.999975  ...   1.527024  17.901435 -2.211148
        4  0.249997  0.249997  0.142855  ...   2.247266  10.012977  9.257347

        [5 rows x 794 columns]
```

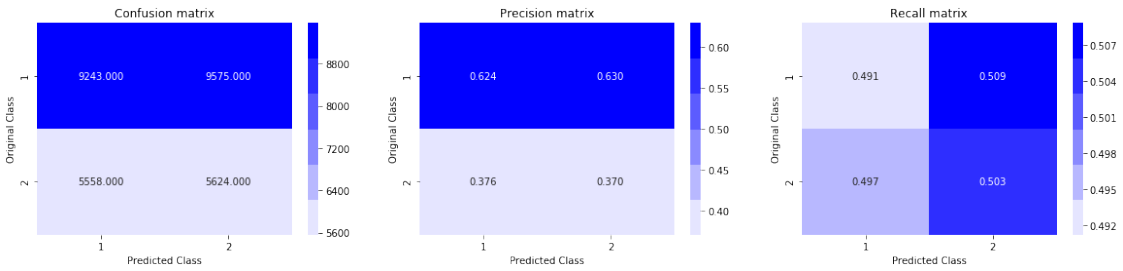
```
In [0]: print(X_train.shape)
        print(y_train.shape)
        print(X_test.shape)
        print(y_test.shape)
```

```
(70000, 794)
(70000,)
(30000, 794)
(30000,)
```

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

```
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
        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, eps=1e-15

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

```
Log loss on Test Data using Random Model 0.8945301761847565
```



#

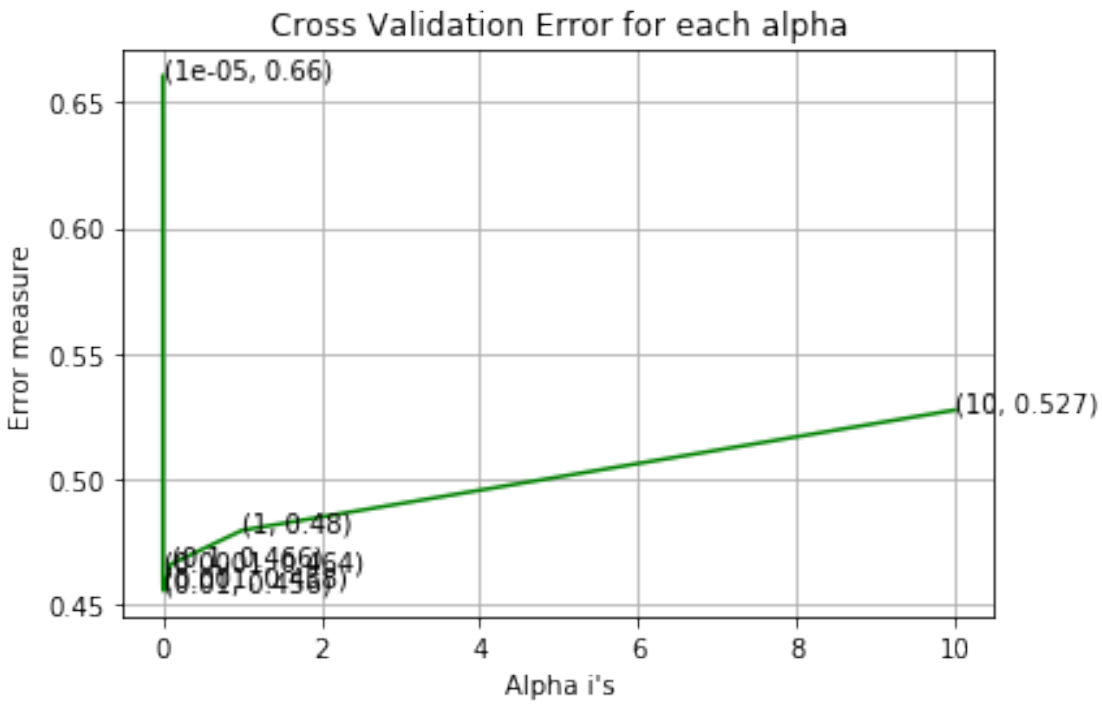
4.4 Logistic Regression with hyperparameter tuning Using **TFIDF-w2v**

```
In [0]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

        # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
        # ------------------------------
        # default parameters
        # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
        # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
        # 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 Gr
```

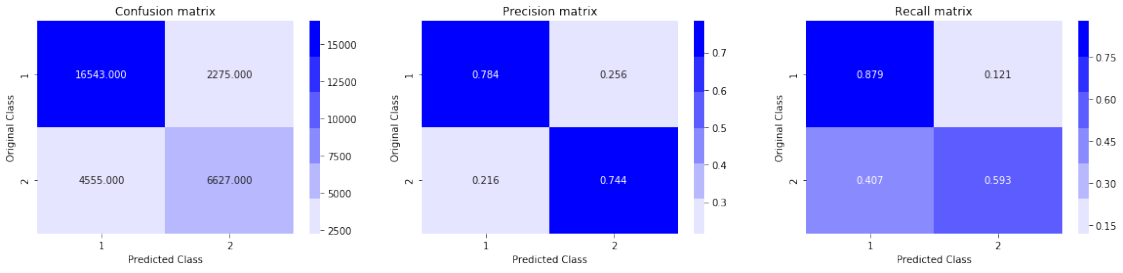
```
        # predict(X)       Predict class labels for samples in X.

        #--------------------------------
        # 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_, eps=1e-15))
            print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, la

        fig, ax = plt.subplots()
        ax.plot(alpha, log_error_array,c='g')
        for i, txt in enumerate(np.round(log_error_array,3)):
            ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
        plt.grid()
        plt.title("Cross Validation Error for each alpha")
        plt.xlabel("Alpha i's")
        plt.ylabel("Error measure")
        plt.show()


        best_alpha = np.argmin(log_error_array)
        clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
        clf.fit(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:",log_los
        predict_y = sig_clf.predict_proba(X_test)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
        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: 0.6603945477700923
For values of alpha =  0.0001 The log loss is: 0.4638402666572263
For values of alpha =  0.001 The log loss is: 0.4579559328978093
For values of alpha =  0.01 The log loss is: 0.4555400335302198
For values of alpha =  0.1 The log loss is: 0.46611619770775015
For values of alpha =  1 The log loss is: 0.47958283736021623
```

```
For values of alpha =  10 The log loss is: 0.5274478061842762
```




```
For values of best alpha =  0.01 The train log loss is: 0.4430192002674681
For values of best alpha =  0.01 The test log loss is: 0.4555400335302198
Total number of data points : 30000
```



#

4.Linear-SVM with hyperparameter tuning Using **TFIDF-w2v**

```
In [0]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

        # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
```

```
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
# 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 Gr
# 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,class_weigh
    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_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, la

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='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_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_los
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
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: 0.6603945477700923
For values of alpha =  0.0001 The log loss is: 0.6603945477700923
For values of alpha =  0.001 The log loss is: 0.6603945477700923
For values of alpha =  0.01 The log loss is: 0.6603945477700923
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
```

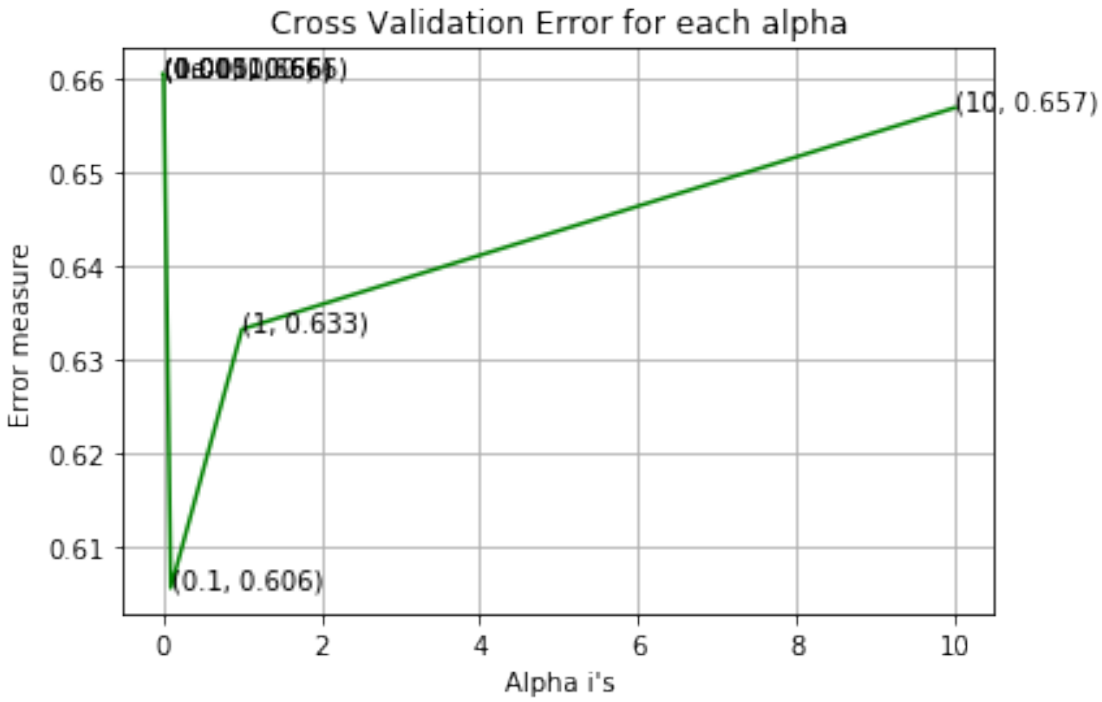
```
For values of alpha =  0.1 The log loss is: 0.6057157277126638
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
```

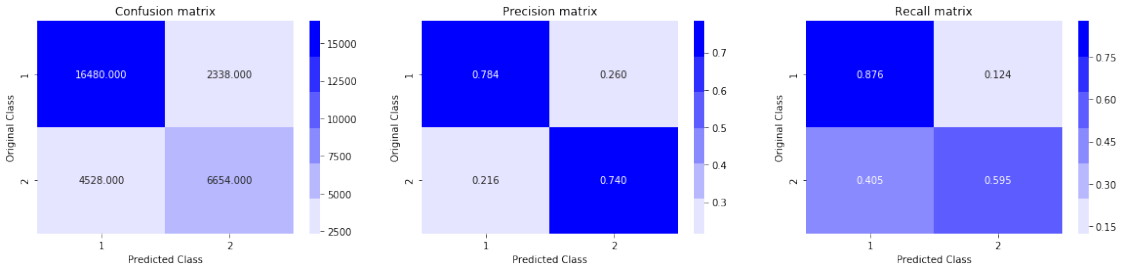
```
For values of alpha =  1 The log loss is: 0.6332415634740302
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:561: Converge
  ConvergenceWarning)
```

```
For values of alpha =  10 The log loss is: 0.6567916612121009
```




```
For values of best alpha =  0.1 The train log loss is: 0.4440709754709528
For values of best alpha =  0.1 The test log loss is: 0.4560399384608992
Total number of data points : 30000
```



4.6 XGBoost using TFIDF-W2V

```
In [0]: import xgboost as xgb
        params = {}
        params['objective'] = 'binary:logistic'
        params['eval_metric'] = 'logloss'
        params['eta'] = 0.02
        params['max_depth'] = 4
```

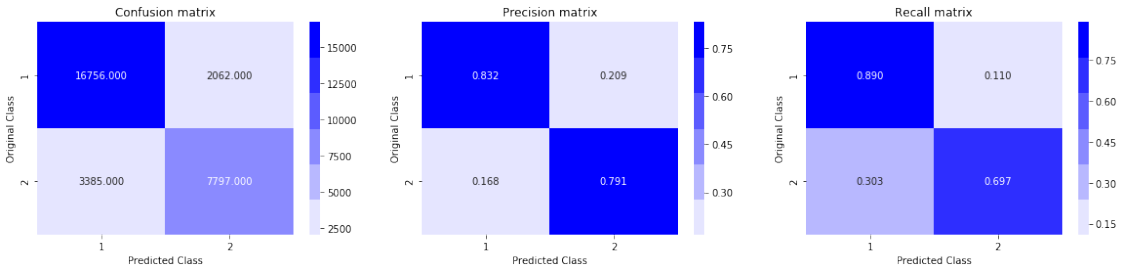
```
        d_train = xgb.DMatrix(X_train, label=y_train)
        d_test = xgb.DMatrix(X_test, label=y_test)

        parameters = {
            'max_depth': [6,7,10,15],
            'min_child_weight':[ 1,5,7],
            'eta':[.1,.3,.6],
            'subsample':[ 1,2,3,4],
            'colsample_bytree': [1,4,7],
            'objective': ['reg:linear'],
        }
        watchlist = [(d_train, 'train'), (d_test, 'valid')]

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

        from sklearn.model_selection import RandomizedSearchCV
        rf_random = RandomizedSearchCV(bst, param_distributions=parameters,
                                            n_iter=10,cv=10,scoring='f1',random_state=25)
        rf_random.fit(d_train,y_train)

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

```
[0]        train-logloss:0.684819        valid-logloss:0.684955
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:0.616095        valid-logloss:0.617449
[20]        train-logloss:0.565241        valid-logloss:0.567207
[30]        train-logloss:0.527422        valid-logloss:0.529905
[40]        train-logloss:0.497954        valid-logloss:0.500989
[50]        train-logloss:0.47499        valid-logloss:0.478488
[60]        train-logloss:0.456586        valid-logloss:0.460536
[70]        train-logloss:0.441758        valid-logloss:0.446063
[80]        train-logloss:0.429758        valid-logloss:0.4344
[90]        train-logloss:0.419838        valid-logloss:0.424843
[100]        train-logloss:0.411643        valid-logloss:0.416991
[110]        train-logloss:0.404705        valid-logloss:0.410402
[120]        train-logloss:0.398948        valid-logloss:0.404953
[130]        train-logloss:0.394175        valid-logloss:0.400506
[140]        train-logloss:0.389694        valid-logloss:0.396312
[150]        train-logloss:0.386048        valid-logloss:0.392926
[160]        train-logloss:0.382934        valid-logloss:0.39012
[170]        train-logloss:0.380088        valid-logloss:0.387575
[180]        train-logloss:0.377546        valid-logloss:0.385292
[190]        train-logloss:0.375311        valid-logloss:0.383323
[200]        train-logloss:0.373236        valid-logloss:0.381474
```

```
[210]        train-logloss:0.370868        valid-logloss:0.379396
[220]        train-logloss:0.368692        valid-logloss:0.377481
[230]        train-logloss:0.366701        valid-logloss:0.375731
[240]        train-logloss:0.364878        valid-logloss:0.374185
[250]        train-logloss:0.363112        valid-logloss:0.372718
[260]        train-logloss:0.361495        valid-logloss:0.371435
[270]        train-logloss:0.360065        valid-logloss:0.370332
[280]        train-logloss:0.358612        valid-logloss:0.369238
[290]        train-logloss:0.357381        valid-logloss:0.368365
[300]        train-logloss:0.356134        valid-logloss:0.367458
[310]        train-logloss:0.355009        valid-logloss:0.366685
[320]        train-logloss:0.353707        valid-logloss:0.36573
[330]        train-logloss:0.352514        valid-logloss:0.36494
[340]        train-logloss:0.351403        valid-logloss:0.364171
[350]        train-logloss:0.350249        valid-logloss:0.363377
[360]        train-logloss:0.34923        valid-logloss:0.362672
[370]        train-logloss:0.34819        valid-logloss:0.362054
[380]        train-logloss:0.347094        valid-logloss:0.361377
[390]        train-logloss:0.346132        valid-logloss:0.360822
[399]        train-logloss:0.345212        valid-logloss:0.360275
The test log loss is: 0.36027528606544557
```

```
In [0]: 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
```



1 TFIDF

```
In [0]: # 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 [0]: df=df[:100000]
        df_train=df[:80000]
        df_test=df[80000:100000]
```

```
In [9]: df_train.head()
```

```
Out[9]:    id  qid1  ...                                          question2 is_duplicate
        0   0     1  ...  What is the step by step guide to invest in sh...            0
        1   1     3  ...  What would happen if the Indian government sto...            0
        2   2     5  ...  How can Internet speed be increased by hacking...            0
        3   3     7  ...  Find the remainder when [math]23^{24}[/math] i...            0
        4   4     9  ...             Which fish would survive in salt water?            0

        [5 rows x 6 columns]
```

```
In [10]: from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.feature_extraction.text import CountVectorizer


         tfidf = TfidfVectorizer(lowercase=False, )

         tfidf_q1_train =tfidf.fit_transform(df_train['question1'])
         tfidf_q1_test  =tfidf.transform(df_test['question1'])
         print(tfidf_q1_train.shape,tfidf_q1_test.shape)

         tfidf = TfidfVectorizer(lowercase=False, )


         tfidf_q2_train =tfidf.fit_transform(df_train['question2'])
         tfidf_q2_test  =tfidf.transform(df_test['question2'])
         print(tfidf_q2_train.shape,tfidf_q2_test.shape)


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

```
(80000, 39323) (20000, 39323)
(80000, 36831) (20000, 36831)
```

```
In [0]: df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
        df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

```
df3 = df_train.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df4 = df_test.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)


df1_train=df1[:80000]
df1_test=df1[80000:100000]
df2_train=df2[:80000]
df2_test=df2[80000:100000]
```

In [24]:
```
y_train.shape
```

Out[24]: (80000,)

In [0]:
```
y_train=df1_train['is_duplicate']
y_test=df1_test['is_duplicate']
```

In [13]:
```
print("Number of features in nlp dataframe :", df1_train.shape)
print("Number of features in preprocessed dataframe :", df2_train.shape)
print("Number of features in question1 w2v  dataframe :", tfidf_q1_train.shape)
print("Number of features in question2 w2v  dataframe :", tfidf_q2_train.shape)
print("Number of features in final train dataframe  :", df1_train.shape[1]+df2_train.sh

print("Number of features in nlp dataframe :", df1_test.shape)
print("Number of features in preprocessed dataframe :", df2_test.shape)
print("Number of features in question1 w2v  dataframe :", tfidf_q2_test.shape)
print("Number of features in question2 w2v  dataframe :", tfidf_q2_test.shape)
print("Number of features in final  train dataframe  :", df1_test.shape[1]+df2_test.sha
```

```
Number of features in nlp dataframe : (80000, 16)
Number of features in preprocessed dataframe : (80000, 12)
Number of features in question1 w2v  dataframe : (80000, 39323)
Number of features in question2 w2v  dataframe : (80000, 36831)
Number of features in final train dataframe  : 76182
Number of features in nlp dataframe : (20000, 16)
Number of features in preprocessed dataframe : (20000, 12)
Number of features in question1 w2v  dataframe : (20000, 36831)
Number of features in question2 w2v  dataframe : (20000, 36831)
Number of features in final  train dataframe  : 76182
```

In [14]:
```
df1_train.head()
```

Out[14]:

	id	cwc_min	cwc_max	...	fuzz_ratio	fuzz_partial_ratio	longest_substr_ratio
0	0	0.999980	0.833319	...	93	100	0.982759
1	1	0.799984	0.399996	...	66	75	0.596154
2	2	0.399992	0.333328	...	54	54	0.166667
3	3	0.000000	0.000000	...	35	40	0.039216
4	4	0.399992	0.199998	...	46	56	0.175000

[5 rows x 16 columns]

```
In [0]: from scipy.sparse import hstack
        train_comb=hstack([tfidf_q1_train,tfidf_q2_train,df1_train,df2_train])
        test_comb=hstack([tfidf_q1_test,tfidf_q2_test,df1_test,df2_test])

In [0]: from sklearn.preprocessing import StandardScaler
        trans=StandardScaler(with_mean=False)
        stand_train=trans.fit_transform(train_comb)
        stand_test=trans.transform(test_comb)

In [0]: X_train=stand_train
        X_test=stand_test
        # y_train=df1_train['is_duplicate']
        # y_test=df1_test['is_duplicate']

In [18]: print("Number of data points in train data :",X_train.shape)
         print("Number of data points in test data :",X_test.shape)

Number of data points in train data : (80000, 76182)
Number of data points in test data : (20000, 76182)


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])/train_
         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])/test_len)

---------- Distribution of output variable in train data ----------
Class 0:  0.627475 Class 1:  0.372525
---------- Distribution of output variable in train data ----------
Class 0:  0.3726 Class 1:  0.3726


In [0]: # This function plots the confusion matrices given y_i, y_i_hat.
        def plot_confusion_matrix(test_y, predict_y):
            C = confusion_matrix(test_y, predict_y)
            # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predict

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

            # C = [[1, 2],
            #      [3, 4]]
            # C.T = [[1, 3],
            #        [2, 4]]
            # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in tu
```

```
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in tu
    # 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, yticklabels=lab
    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, yticklabels=lab
    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, yticklabels=lab
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

#

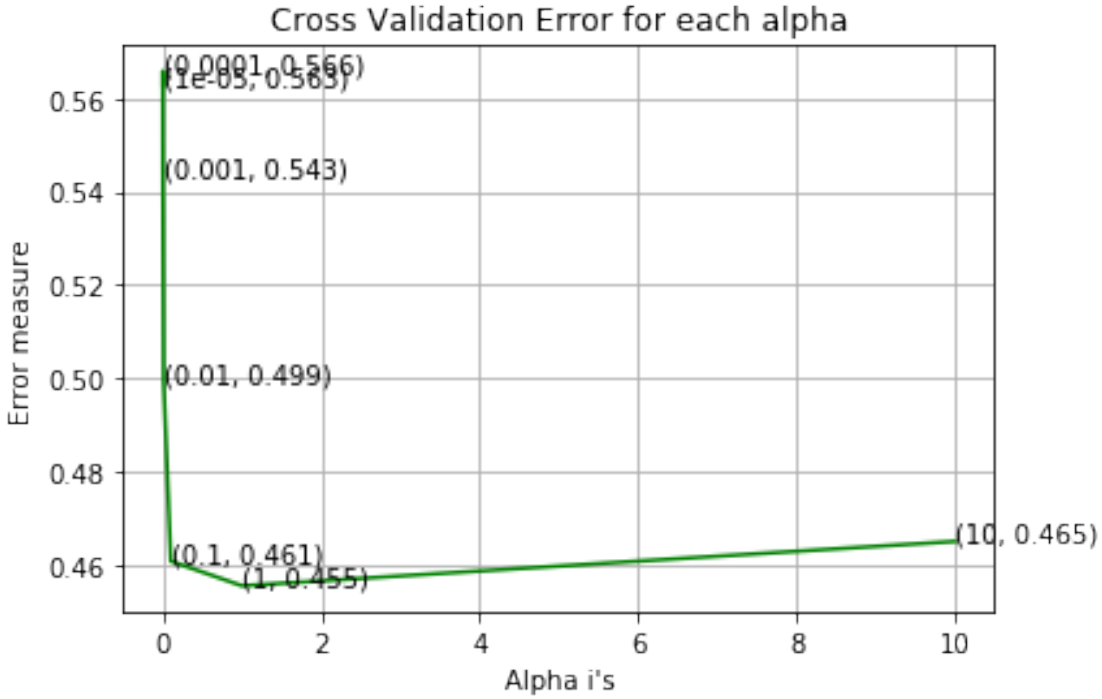
4.4 Logistic Regression with hyperparameter tuning Using **TFIDF**

```
In [30]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

        # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
```

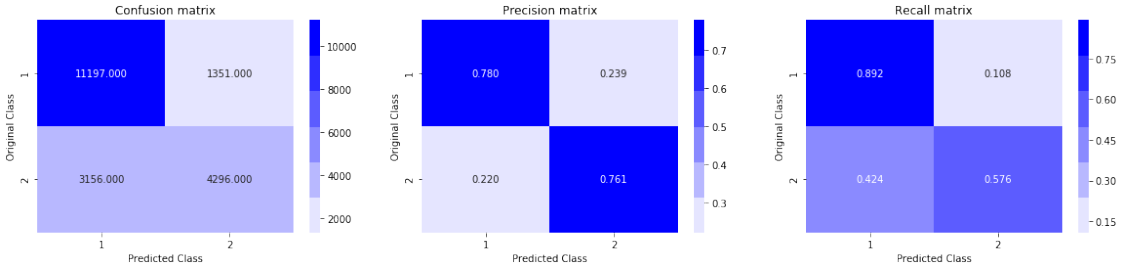
```
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# class_weight=None, warm_start=False, average=False, n_iter=None)

# 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,class_weight
    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_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, l

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42,
clf.fit(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:",log_lo
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
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: 0.5627967130099867
For values of alpha =  0.0001 The log loss is: 0.5659275273184717
For values of alpha =  0.001 The log loss is: 0.5434918599838902
For values of alpha =  0.01 The log loss is: 0.4993338211142962
For values of alpha =  0.1 The log loss is: 0.46074361888842585
For values of alpha =  1 The log loss is: 0.4554505018083464
For values of alpha =  10 The log loss is: 0.46496426056827234
```




```
For values of best alpha =  1 The train log loss is: 0.4195807597822445
For values of best alpha =  1 The test log loss is: 0.4554505018083464
Total number of data points : 20000
```

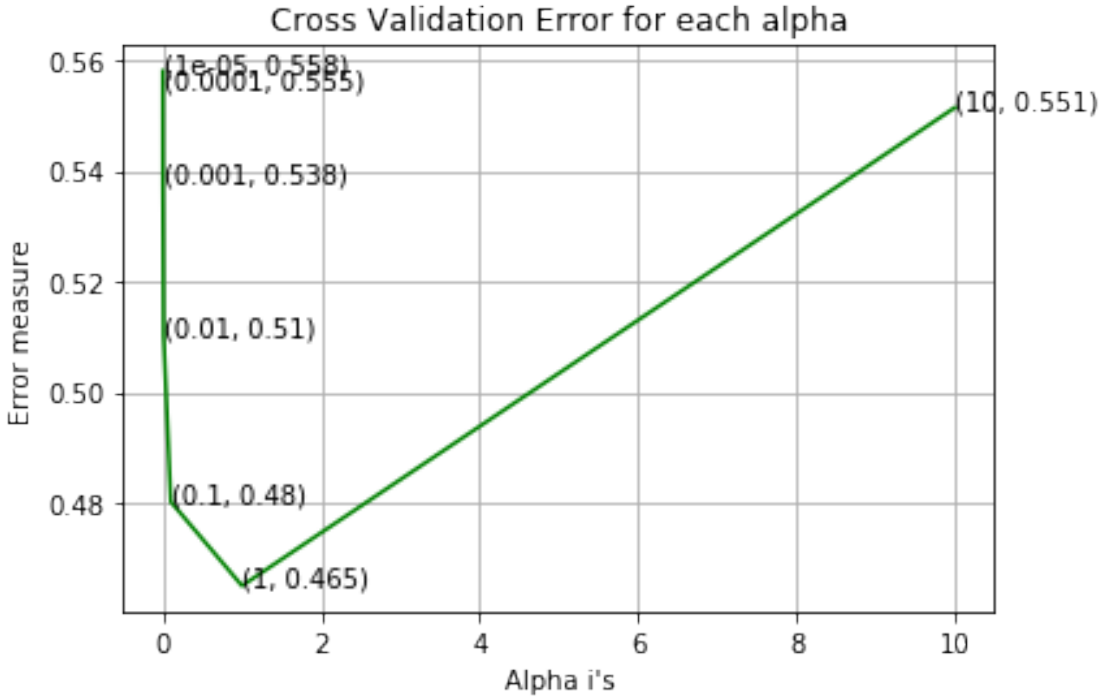


2 LINEAR-SVM using tf-idf

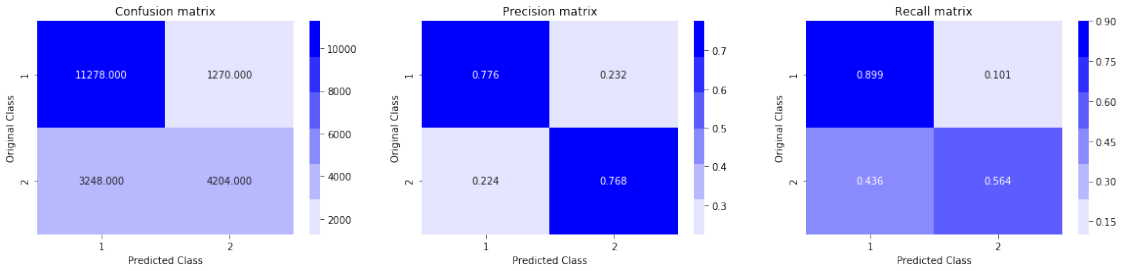
In [31]:
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/s
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opt
# 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='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_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, l

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='hinge', random_state=4
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:",.10+lo
        predict_y = sig_clf.predict_proba(X_test)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
        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: 0.5580805854625569
For values of alpha =  0.0001 The log loss is: 0.5550108929226976
For values of alpha =  0.001 The log loss is: 0.5380308126106796
For values of alpha =  0.01 The log loss is: 0.510139205555509
For values of alpha =  0.1 The log loss is: 0.48016037380011256
For values of alpha =  1 The log loss is: 0.4649376078033968
For values of alpha =  10 The log loss is: 0.5514451797543597
```




```
For values of best alpha =  1 The train log loss is: 0.4069894832508393
For values of best alpha =  1 The test log loss is: 0.4649376078033968
Total number of data points : 20000
```



4.6 XGBoost using TFIDF

```
In [0]: print(X_train.shape)
        print(y_train.shape)
        print(X_test.shape)
        print(y_test.shape)

(80000, 76182)
(80000,)
(20000, 76182)
(20000,)
```

```
In [32]: 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)

         parameters = {
             'max_depth': [6,7,10,15],
             'min_child_weight':[ 1,5,7],
             'eta':[.1,.3,.6],
             'subsample':[ 1,2,3,4],
             'colsample_bytree': [1,4,7],
             'objective': ['reg:linear'],
         }
         watchlist = [(d_train, 'train'), (d_test, 'valid')]

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

         from sklearn.model_selection import RandomizedSearchCV
         rf_random = RandomizedSearchCV(bst, param_distributions=parameters,
                                        n_iter=10,cv=10,scoring='f1',random_state=25)
```

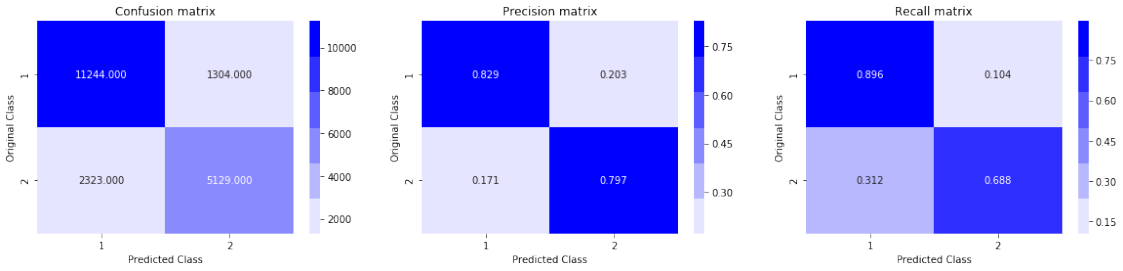
```
        rf_random.fit(d_train,y_train)
        #
        xgdmat = xgb.DMatrix(d_train,y_train)
        print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-1
        predict_y = bst.predict(d_test)
```

```
[0]        train-logloss:0.684834        valid-logloss:0.684968
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:0.616281        valid-logloss:0.617535
[20]        train-logloss:0.565253        valid-logloss:0.567295
[30]        train-logloss:0.527287        valid-logloss:0.529942
[40]        train-logloss:0.49765        valid-logloss:0.500929
[50]        train-logloss:0.474965        valid-logloss:0.478632
[60]        train-logloss:0.456705        valid-logloss:0.460855
[70]        train-logloss:0.441911        valid-logloss:0.446439
[80]        train-logloss:0.429958        valid-logloss:0.434904
[90]        train-logloss:0.420205        valid-logloss:0.425473
[100]        train-logloss:0.411937        valid-logloss:0.417508
[110]        train-logloss:0.405121        valid-logloss:0.410958
[120]        train-logloss:0.399333        valid-logloss:0.405359
[130]        train-logloss:0.394379        valid-logloss:0.400567
[140]        train-logloss:0.39008        valid-logloss:0.39643
[150]        train-logloss:0.386257        valid-logloss:0.39276
[160]        train-logloss:0.383225        valid-logloss:0.389988
[170]        train-logloss:0.380494        valid-logloss:0.3874
[180]        train-logloss:0.378101        valid-logloss:0.385183
[190]        train-logloss:0.375911        valid-logloss:0.383171
[200]        train-logloss:0.37403        valid-logloss:0.381441
[210]        train-logloss:0.372269        valid-logloss:0.379873
[220]        train-logloss:0.370751        valid-logloss:0.378523
[230]        train-logloss:0.369222        valid-logloss:0.377176
[240]        train-logloss:0.367528        valid-logloss:0.375693
[250]        train-logloss:0.365985        valid-logloss:0.374392
[260]        train-logloss:0.364475        valid-logloss:0.373031
[270]        train-logloss:0.363256        valid-logloss:0.371974
[280]        train-logloss:0.362112        valid-logloss:0.370971
[290]        train-logloss:0.361066        valid-logloss:0.370089
[300]        train-logloss:0.360172        valid-logloss:0.369339
[310]        train-logloss:0.359131        valid-logloss:0.368448
[320]        train-logloss:0.358202        valid-logloss:0.367693
[330]        train-logloss:0.357338        valid-logloss:0.366961
[340]        train-logloss:0.356504        valid-logloss:0.366299
[350]        train-logloss:0.355703        valid-logloss:0.365669
[360]        train-logloss:0.355017        valid-logloss:0.365093
[370]        train-logloss:0.354307        valid-logloss:0.364501
[380]        train-logloss:0.353636        valid-logloss:0.363949
```

```
[390]        train-logloss:0.352976        valid-logloss:0.36341
[399]        train-logloss:0.352416        valid-logloss:0.362957
The test log loss is: 0.4649376078033968
```

```
In [33]: 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 : 20000
```




```
In [0]: train_comb.to_csv('train_comb.csv')
        test_comb.to_csv('test_comb.csv')
```

5. CONCLUSION

1) Loadedthe data
2) Imported the libraries
3) Having two pair of question which we have find if there meaning is similar
4) Removing all the dublicates question pairs
5) Removing all the null values
6) 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

7) analysing the pdf of word-share and word common

8)Preprocessing the text from the columns pf quesions like removing stop-words,punctuations,performing stemming

9)Addiing advanced feature extraction

10)Making words clouds to understand most used words

11)Ploting pair plots of these new features extracted

12)Plotting TSNE which reduces the dimention and give a 2d-view

13) Splitted the dataframe into test and train and then Converting the text of question1 and question2 into vector form using

1) **TFIDF-W2V**

a) merged all the features into one dataset and then applied
1)logistic classifier - train_log_loss= .44 , test_log_loss = .45
2)linear svm - train_log_loss = .44, test_log_loss= .45
3)xgbooost - train_log_loss = 0.34 test_log_loss=.36

2) **TFIDF**

a) hstack all the features into one dataset then standerdised the data and then applied
1)logistic classifier - train_log_loss= 0.41, test_log_loss= .45
2)linear svm - train_log_loss= .40, test_log_loss= 0.46
3)xgbooost - train_log_loss= .35 test_log_loss=.36