In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
from wordcloud import WordCloud,STOPWORDS
from PIL import Image
import distance
import warnings
warnings.filterwarnings("ignore")
```

```
D:\Anaconda\envs\tensorflow\lib\site-packages\fuzzywuzzy\fuzz.py:35: UserWarnin
g: Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove
this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levensh
tein to remove this warning')
```

In [2]:

```
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('')
```

In [3]:

```
df.head(2)
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_wc
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	

Preprocessing of Text

preprocessing:

```
Removing html tags
Removing Punctuations
Performing stemming
Removing Stopwords
Expanding contractions etc.
```

In [4]:

```
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 n
                           .replace("n't", " not").replace("what's", "what is").r
                           .replace("'ve", " have").replace("i'm", "i am").replac
                           .replace("he's", "he is").replace("she's", "she is").r
                           .replace("%", " percent ").replace("₹", " 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
```

Advanced Feature Extraction(NLP and Fuzzy Features)

In [5]:

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

    #Getting the non stop_words
    q1_stopwords = set([word for word in q1_tokens if word  in STOP_Words])
    q2_stopwords = set([word for word in q2_tokens if word  in STOP_Words])

    #Getting the stop_words
    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])

    common_word_count  =  len(q1_words.intersection(q2_words))

    common_stop_count  =   len(q1_stopwords.intersection(q2_stopwords))

    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_
    token_features[1] = common_word_count/(max(len(q1_words),len(q2_words))+SAFE_
    token_features[2] = common_stop_count/(min(len(q1_stopwords),len(q2_stopwords
    token_features[3] = common_stop_count/(max(len(q1_stopwords),len(q2_stopwords
    token_features[4] = common_token_count/(min(len(q1_tokens),len(q2_tokens)))
    token_features[5] = common_token_count/(max(len(q1_tokens),len(q2_tokens)))

    #last word of both the questions should be the same
    token_features[6] = int(q1_tokens[-1] == q2_tokens[-1])

    #first word of both the 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))

    #Avergae Token len of both questions
    token_features[9] = (len(q1_tokens)+len(q2_tokens))/2
    return token_features
```

In [6]:

```
#getting the logest substring

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)
```

In [7]:

```
def extract_features(df):
    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["que

    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

    print("Fuzzy Features....")

    df['token_set_ratio']   = df.apply(lambda x: fuzz.token_set_ratio(x['question
    df['token_sort_ratio']  = df.apply(lambda x: fuzz.token_sort_ratio(x['questio
    df['fuzz_ratio']        = df.apply(lambda x: fuzz.QRatio(x['question1'],x['qu
    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['q
    return df
```

In [8]:

```
# if os.path.isfile('nlp_features_train.csv'):
#     df = pd.read_csv('nlp_features_train.csv',encoding = 'latin-1')
#     df.fillna('')
# else:
df = pd.read_csv('train.csv')
df = extract_features(df)
```

```
token features....
Fuzzy Features....
```

In [9]:

```
df.head(2)
```

Out[9]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983	...
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988	...

2 rows × 21 columns

In [10]:

```
df.to_csv('NLP_features_train.csv',index = False)
```

Analysis of extracted features

In [11]:

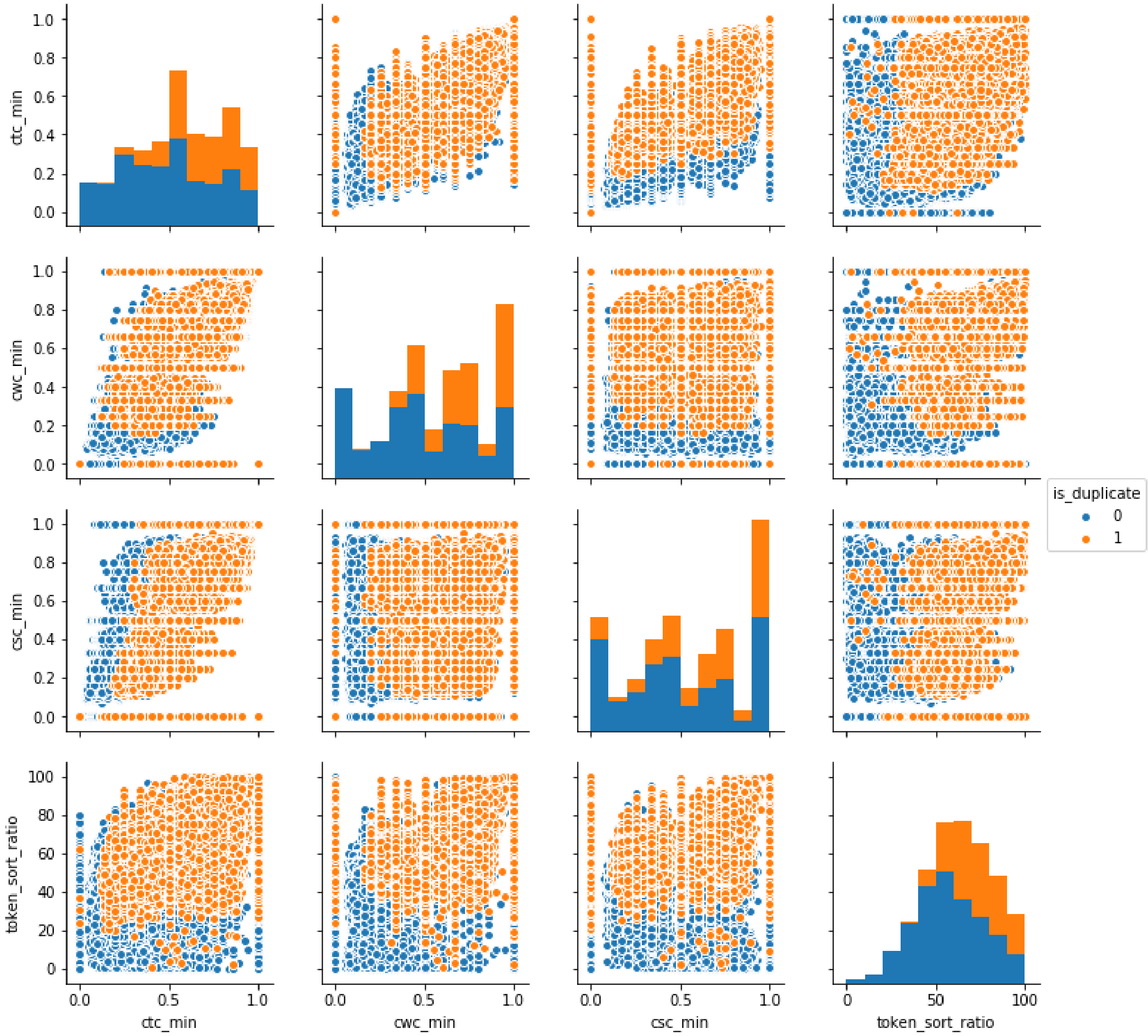
```
df_duplicates = df[df['is_duplicate']==1]
df_nonduplicates = df[df['is_duplicate']==0]

p = np.dstack([df_duplicates['question1'],df_duplicates['question2']]).flatten()
n = np.dstack([df_nonduplicates['question1'],df_nonduplicates['question2']]).flat
```

In [12]:

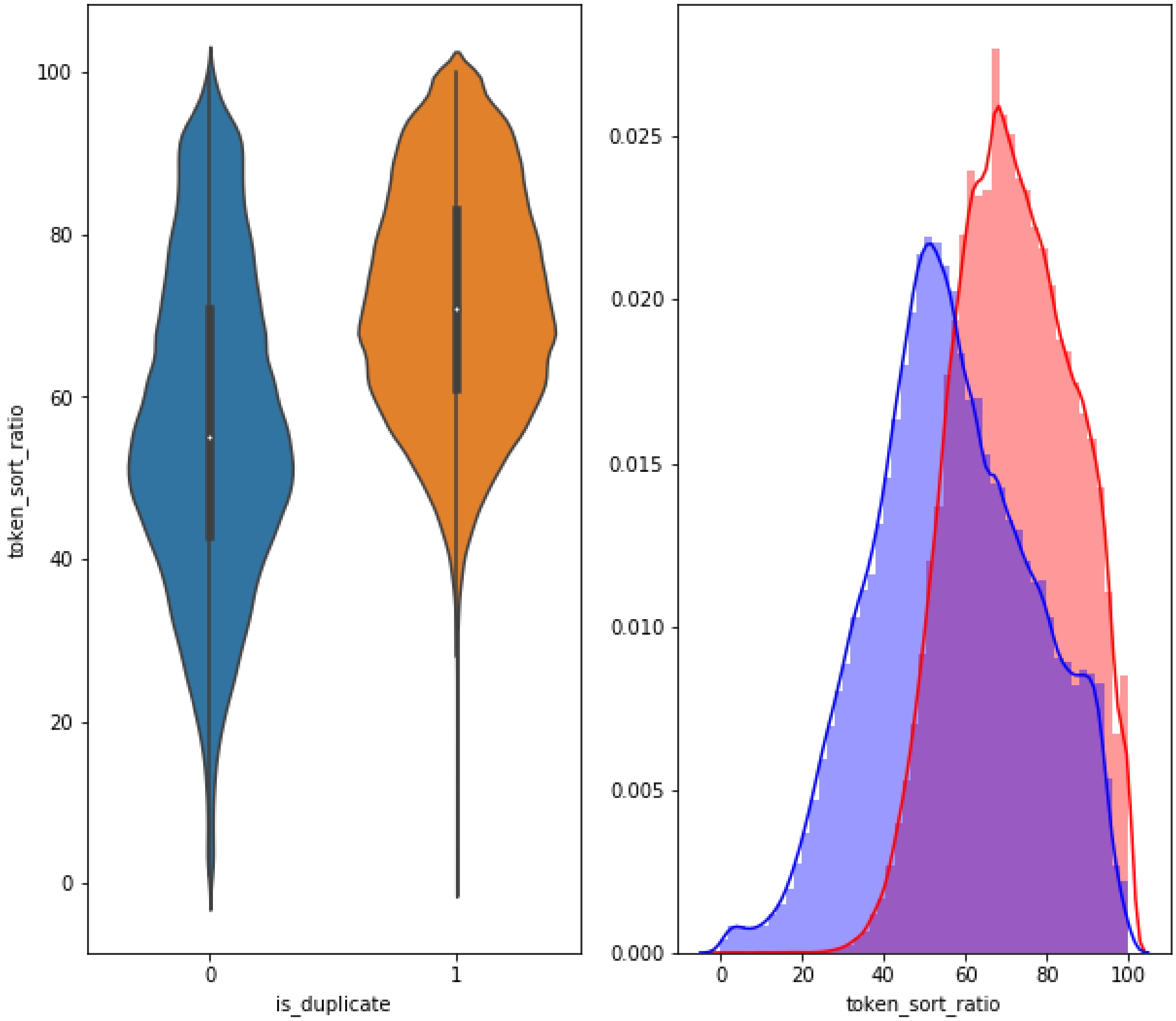
```
print("The number of data points in class1(duplicate pairs):",len(p))
print("The number of data points in class0(nonduplicate pairs):",len(n))
```

```
The number of data points in class1(duplicate pairs): 298526
The number of data points in class0(nonduplicate pairs): 510054
```

In [13]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplica
plt.show()
```

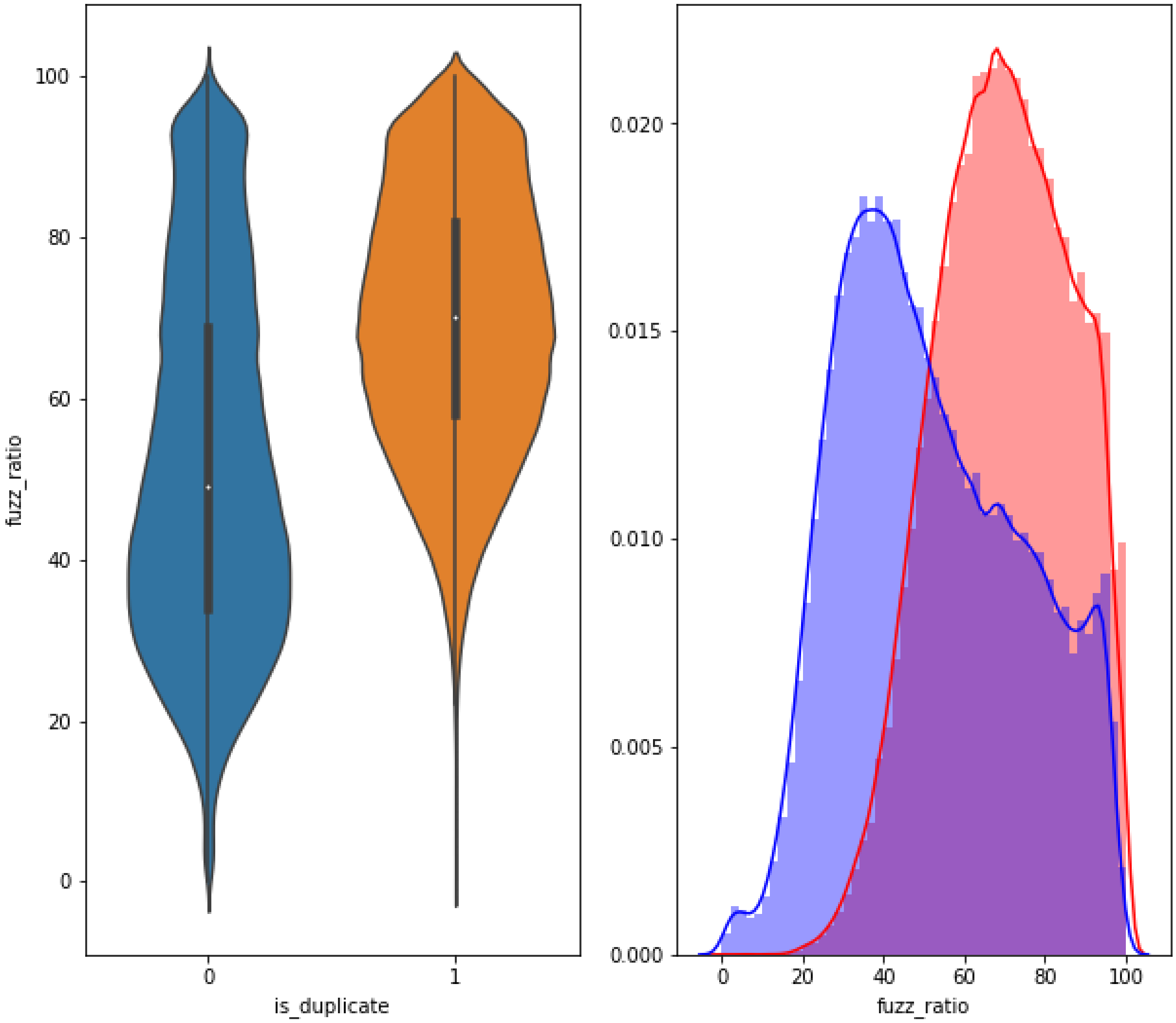


In [14]:
```
plt.figure(figsize=(10,9))

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



In [15]:

```
plt.figure(figsize=(10,9))

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='blue'
plt.show()
```



In [16]:

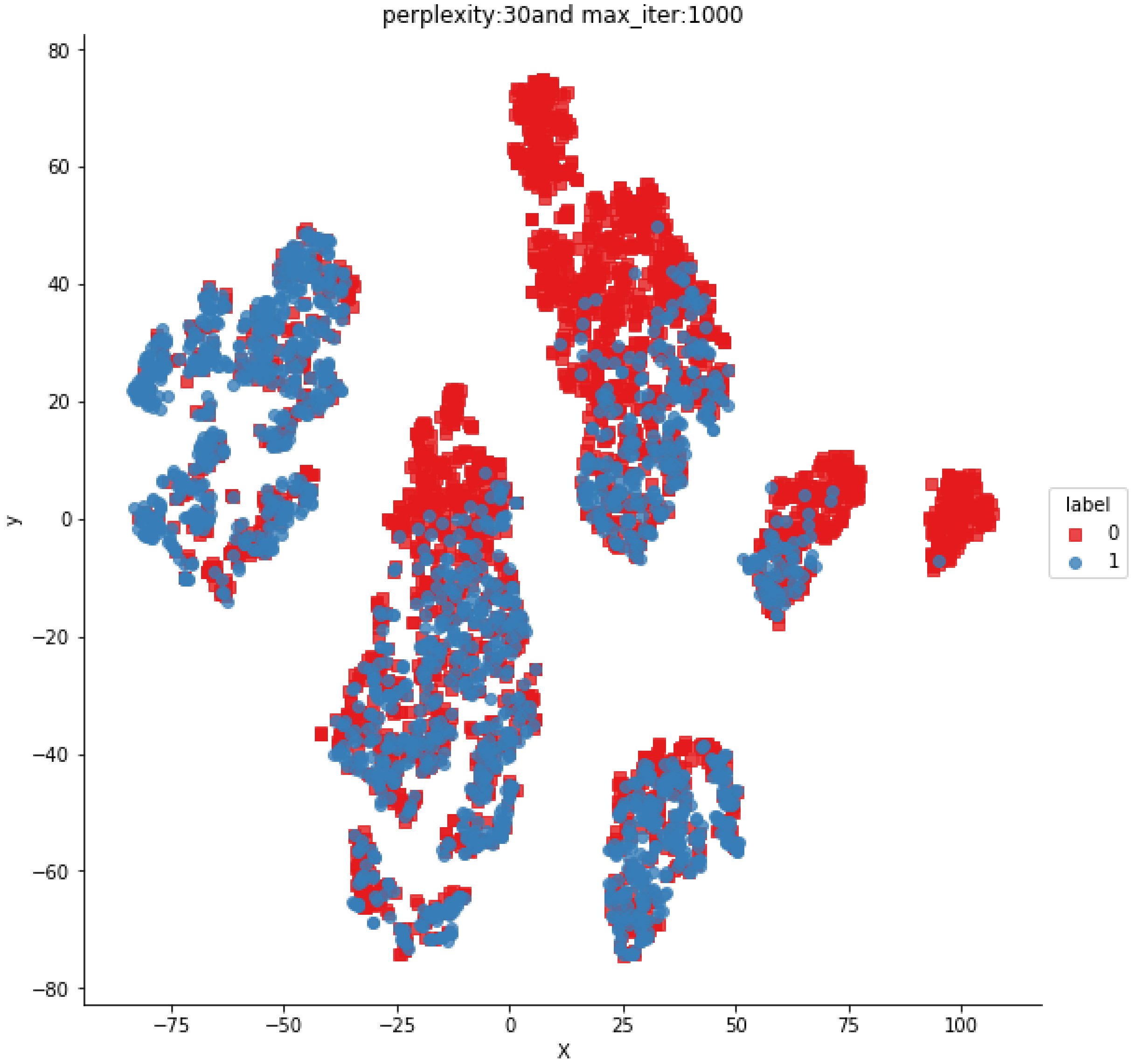
```
from sklearn.preprocessing import MinMaxScaler

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

In [17]:
```
tsen2d = TSNE(n_components=2,init='random',random_state=101,method='barnes_hut',n
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.023s...
[t-SNE] Computed neighbors for 5000 samples in 0.354s...
[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: 0.130446
[t-SNE] Computed conditional probabilities in 0.221s
[t-SNE] Iteration 50: error = 81.2830734, gradient norm = 0.0463674 (50 iterati
ons in 7.061s)
[t-SNE] Iteration 100: error = 70.6134720, gradient norm = 0.0091821 (50 iterat
ions in 4.999s)
[t-SNE] Iteration 150: error = 68.9090500, gradient norm = 0.0058176 (50 iterat
ions in 4.819s)
[t-SNE] Iteration 200: error = 68.0998688, gradient norm = 0.0052551 (50 iterat
ions in 4.985s)
[t-SNE] Iteration 250: error = 67.5875854, gradient norm = 0.0037655 (50 iterat
ions in 5.127s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.587585
[t-SNE] Iteration 300: error = 1.7926569, gradient norm = 0.0011881 (50 iterati
ons in 5.536s)
[t-SNE] Iteration 350: error = 1.3936219, gradient norm = 0.0004811 (50 iterati
ons in 5.598s)
[t-SNE] Iteration 400: error = 1.2277873, gradient norm = 0.0002776 (50 iterati
ons in 5.487s)
[t-SNE] Iteration 450: error = 1.1383334, gradient norm = 0.0001871 (50 iterati
ons in 5.456s)
[t-SNE] Iteration 500: error = 1.0833324, gradient norm = 0.0001436 (50 iterati
ons in 5.677s)
[t-SNE] Iteration 550: error = 1.0474558, gradient norm = 0.0001139 (50 iterati
ons in 5.457s)
[t-SNE] Iteration 600: error = 1.0232476, gradient norm = 0.0000982 (50 iterati
ons in 5.362s)
[t-SNE] Iteration 650: error = 1.0066655, gradient norm = 0.0000856 (50 iterati
ons in 5.443s)
[t-SNE] Iteration 700: error = 0.9949726, gradient norm = 0.0000777 (50 iterati
ons in 5.395s)
[t-SNE] Iteration 750: error = 0.9861333, gradient norm = 0.0000712 (50 iterati
ons in 5.424s)
[t-SNE] Iteration 800: error = 0.9788581, gradient norm = 0.0000651 (50 iterati
ons in 5.440s)
[t-SNE] Iteration 850: error = 0.9731057, gradient norm = 0.0000637 (50 iterati
ons in 5.444s)
[t-SNE] Iteration 900: error = 0.9683460, gradient norm = 0.0000571 (50 iterati
ons in 5.425s)
[t-SNE] Iteration 950: error = 0.9642810, gradient norm = 0.0000567 (50 iterati
ons in 5.425s)
[t-SNE] Iteration 1000: error = 0.9608909, gradient norm = 0.0000565 (50 iterat
ions in 5.441s)
[t-SNE] Error after 1000 iterations: 0.960891
```

In [18]:

```
df = pd.DataFrame({'X':tsen2d[:,0],'y':tsen2d[:,1],'label':y})

sns.lmplot(data=df,x='X',y='y',hue='label',fit_reg=False,size=8,palette='Set1',ma
plt.title("perplexity:{}and max_iter:{}".format(30,1000))
plt.show()
```



In [ ]: