In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image
```

```
C:\Users\ankan\Anaconda3\lib\site-packages\fuzzywuzzy\fuzz.py:11: UserWarning:

Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove thi
s warning
```

In [2]:
```
os.chdir('C:\\Users\\ankan\\Desktop\\Quora')
```

In [3]:
```
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = [10, 5]
warnings.filterwarnings("ignore", category=FutureWarning)
%config InlineBackend.figure_format = 'retina'
```

In [4]:
```
df = pd.read_csv("df_fe_without_preprocessing_train.csv")

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

print('~-> Total number of question pairs for training <-- :\n   {} '.format(len(df
)))

df.head()
```

```
Number of data points: 404290
~-> Total number of question pairs for training <-- :
   404290
```

Out[4]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_n_words
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
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
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
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
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

In [5]:
```
# 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
").replace("can't", "can not")\
                           .replace("n't", " not").replace("what's", "what is").rep
lace("it's", "it is")\
                           .replace("'ve", " have").replace("i'm", "i am").replace(
"'re", " are")\
                           .replace("he's", "he is").replace("she's", "she is").rep
lace("'s", " own")\
                           .replace("%", " percent ").replace("₹", " rupee ").repla
ce("$", " dollar ")\
                           .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
```

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [9]:
```
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)) + SA
FE_DIV)
    token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SA
FE_DIV)
    token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SA
FE_DIV)
    token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SA
FE_DIV)
    token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) +
SAFE_DIV)
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) +
SAFE_DIV)

    # 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)
```

In [11]:
```
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("df_fe_without_preprocessing_train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

Out[11]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	ctc_
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	...	0.78
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	...	0.46

2 rows × 21 columns

In [13]:
```
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"]]).flatt
en()

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))
```

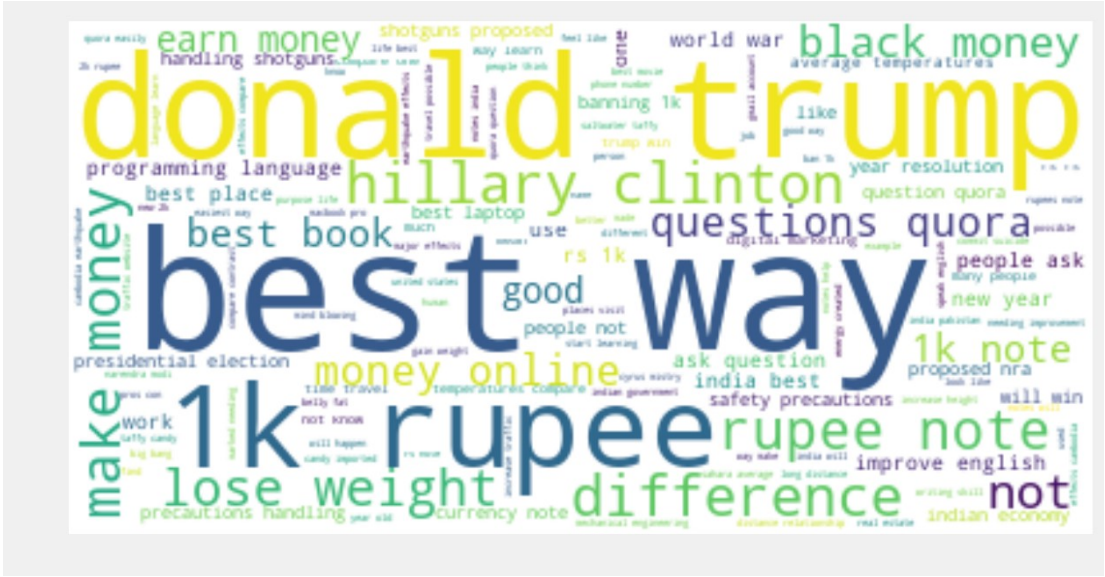
```
Number of data points in class 1 (duplicate pairs) : 298526
Number of data points in class 0 (non duplicate pairs) : 510054
```

In [14]:
```
# 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 : 3335825
```

In [15]:
```
wc = WordCloud(background_color="white", max_words=len(textp_w), stopwords=stopword
s)
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
```

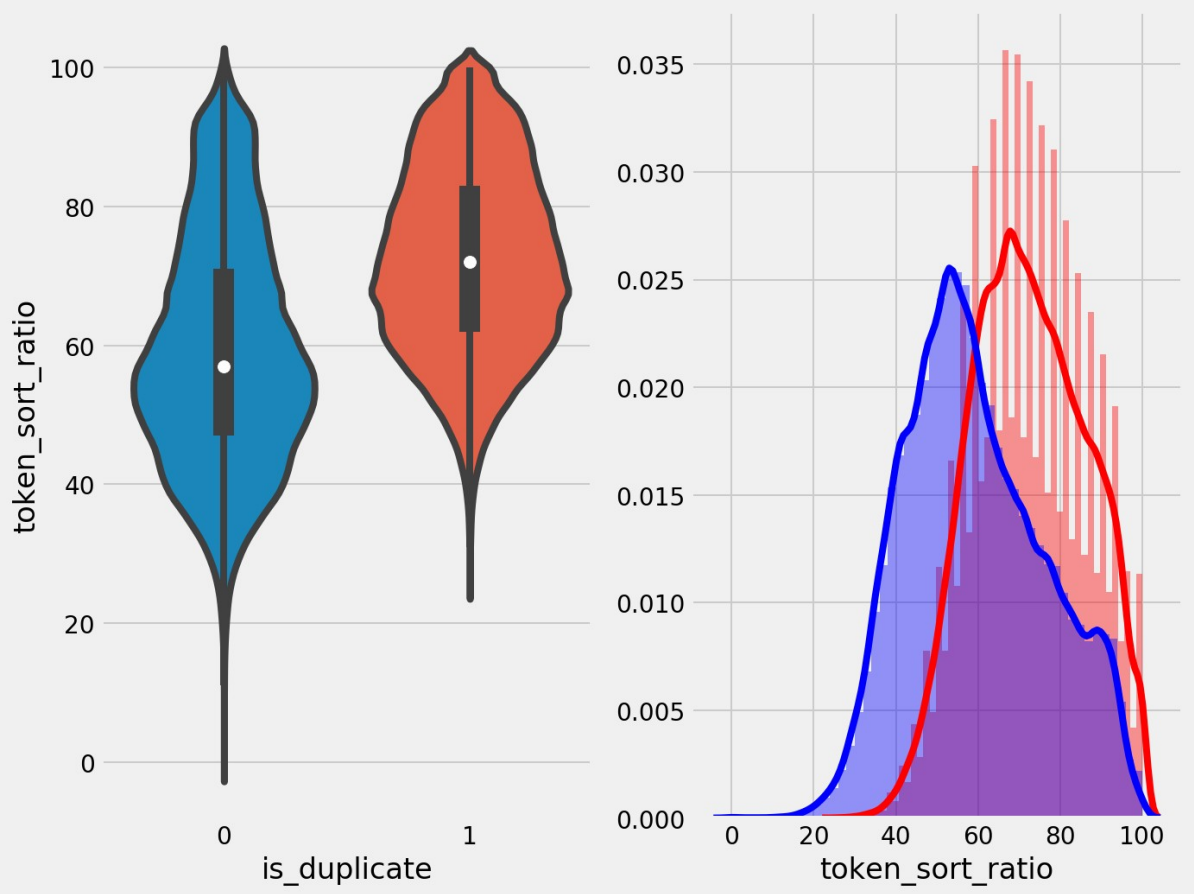


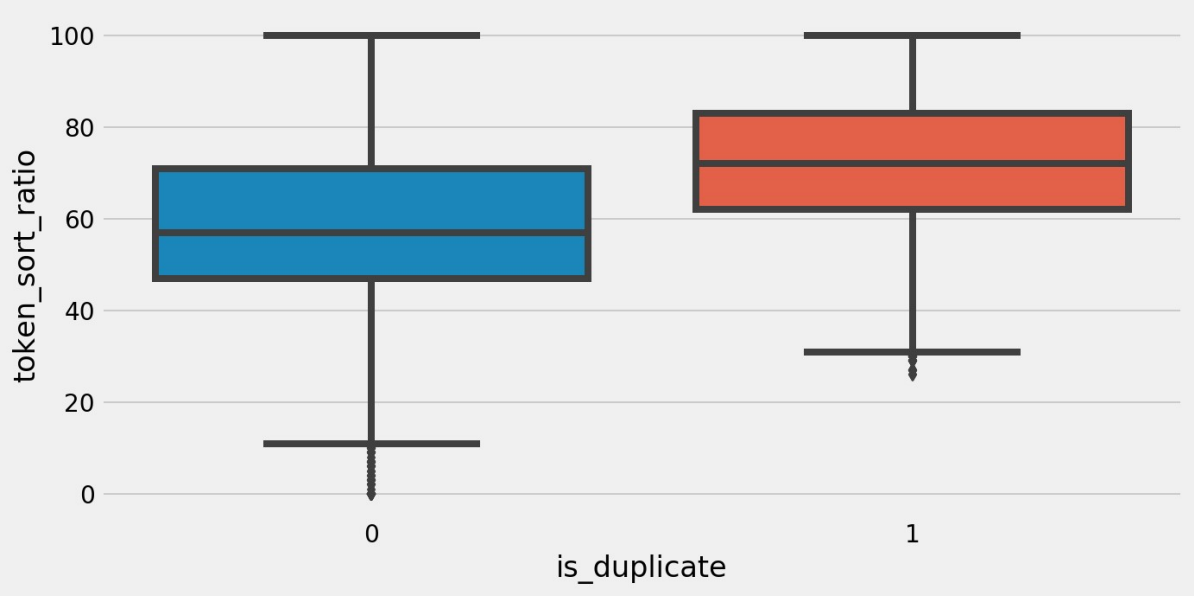
In [16]:
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords
)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

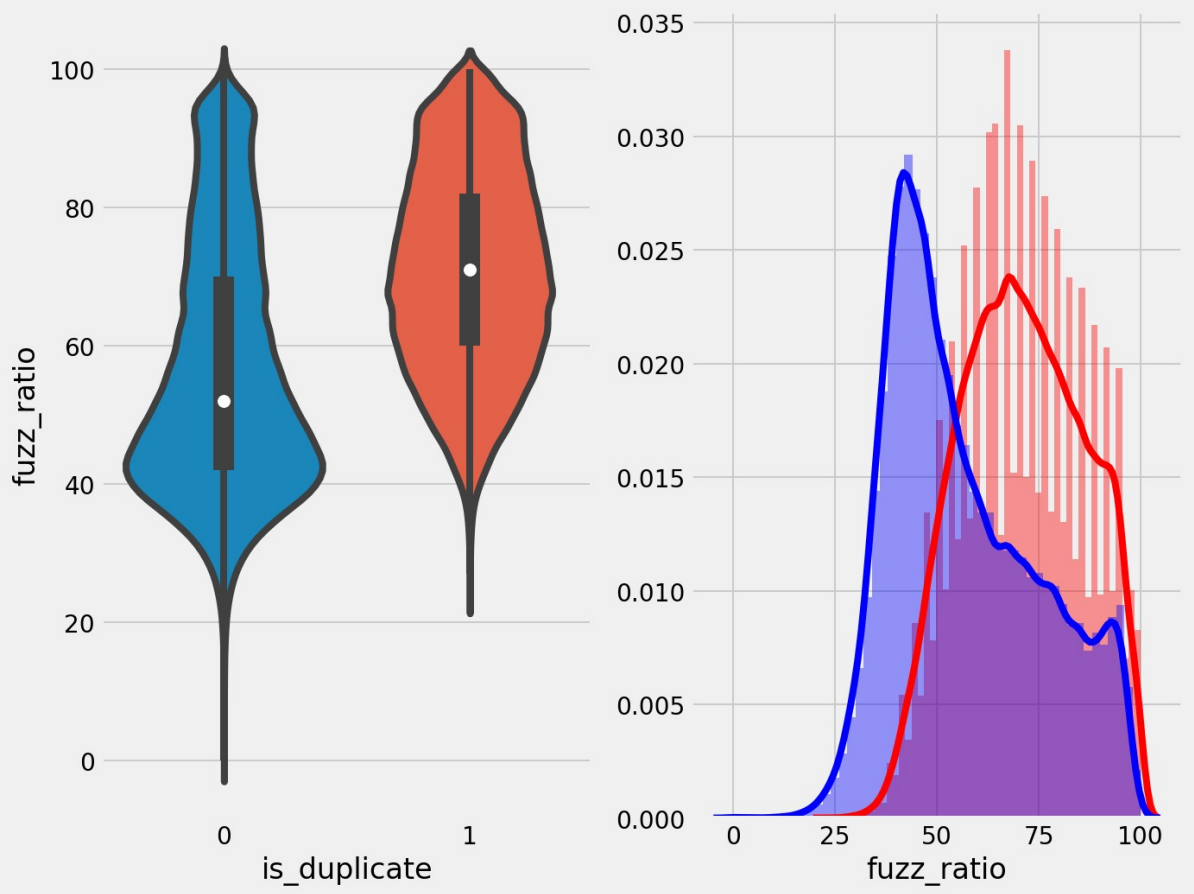
In [17]:
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate
']][0:n], hue='is_duplicate', vars=['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ra
tio'])
plt.show()
```

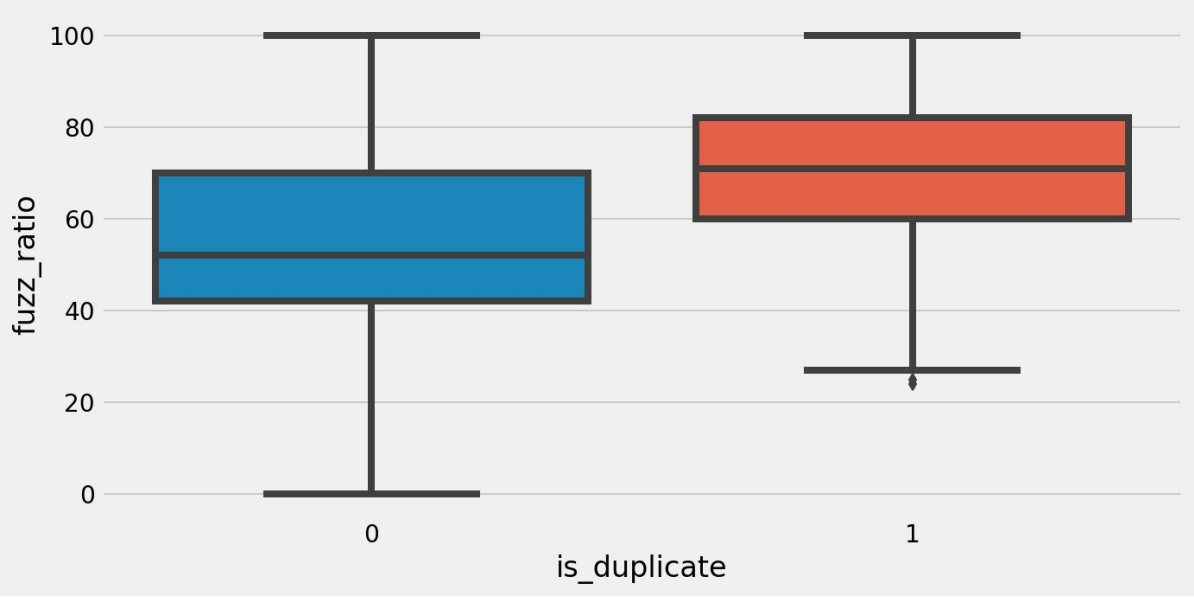
In [32]:
```
import warnings
warnings.filterwarnings("ignore")
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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", c
olor = 'red')
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" ,
color = 'blue' )

plt.show()

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


100
80
60
40
20
0
token_sort_ratio
0
1
is_duplicate
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
token_sort_ratio


100
80
60
40
20
0
token_sort_ratio
0
1
is_duplicate

In [33]:
```
import warnings
warnings.filterwarnings("ignore")
# Distribution of the token_sort_ratio
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
= 'blue' )

plt.show()

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



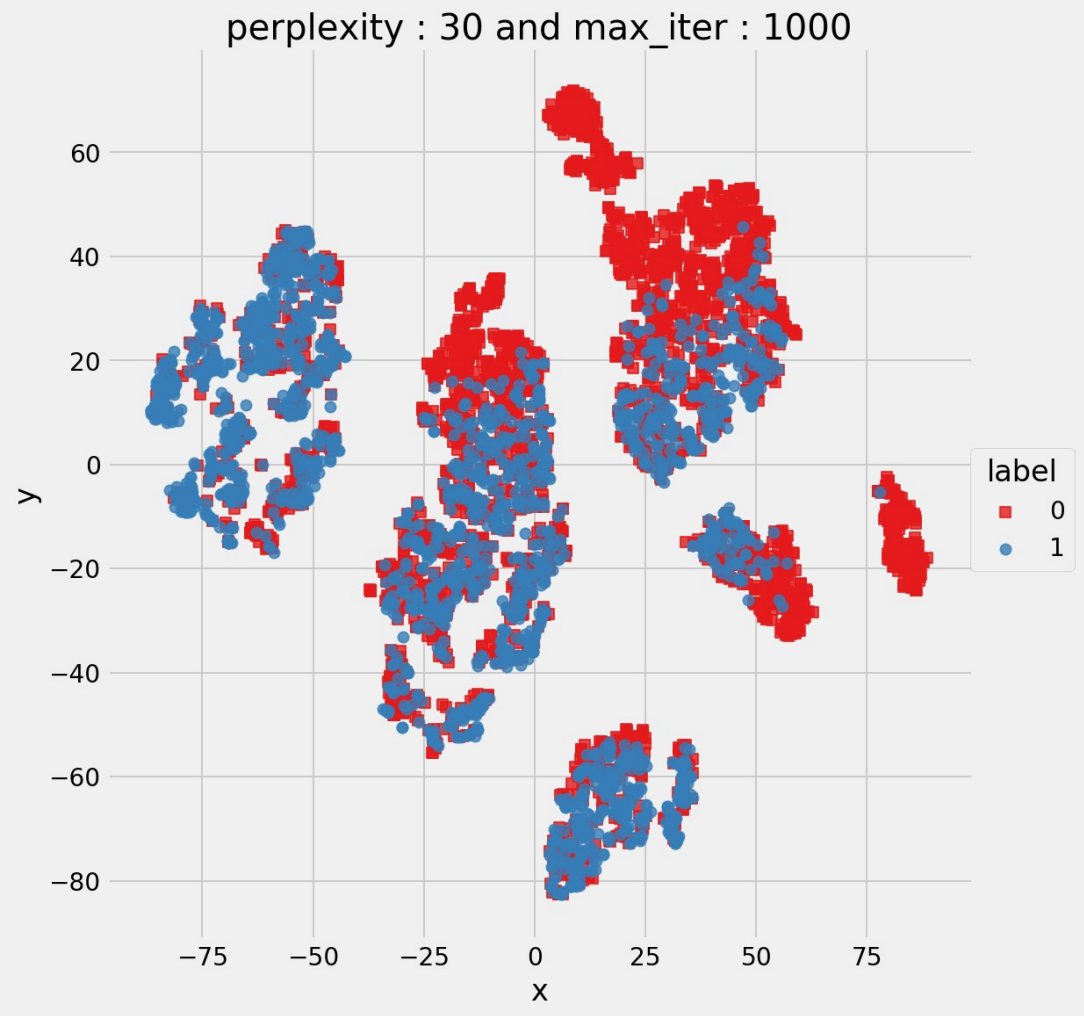


In [34]:
```
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning
the data) to 3 dimention

from sklearn.preprocessing import MinMaxScaler

dfp_subsampled = df[0:5000]
X = MinMaxScaler().fit_transform(dfp_subsampled[['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_parti
al_ratio' , 'longest_substr_ratio']])
y = dfp_subsampled['is_duplicate'].values
```

In [35]:
```
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 0.031s...
[t-SNE] Computed neighbors for 5000 samples in 0.469s...
[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 0.219s
[t-SNE] Iteration 50: error = 80.8968964, gradient norm = 0.0430571 (50 iteratio
ns in 6.514s)
[t-SNE] Iteration 100: error = 70.3833160, gradient norm = 0.0099593 (50 iterati
ons in 5.030s)
[t-SNE] Iteration 150: error = 68.6159134, gradient norm = 0.0056708 (50 iterati
ons in 5.108s)
[t-SNE] Iteration 200: error = 67.7694321, gradient norm = 0.0040581 (50 iterati
ons in 5.264s)
[t-SNE] Iteration 250: error = 67.2746048, gradient norm = 0.0033067 (50 iterati
ons in 5.108s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.274605
[t-SNE] Iteration 300: error = 1.7729300, gradient norm = 0.0011900 (50 iteratio
ns in 5.327s)
[t-SNE] Iteration 350: error = 1.3714967, gradient norm = 0.0004818 (50 iteratio
ns in 5.139s)
[t-SNE] Iteration 400: error = 1.2036748, gradient norm = 0.0002779 (50 iteratio
ns in 5.186s)
[t-SNE] Iteration 450: error = 1.1132656, gradient norm = 0.0001889 (50 iteratio
ns in 5.155s)
[t-SNE] Iteration 500: error = 1.0582460, gradient norm = 0.0001434 (50 iteratio
ns in 5.171s)
[t-SNE] Iteration 550: error = 1.0222589, gradient norm = 0.0001180 (50 iteratio
ns in 5.186s)
[t-SNE] Iteration 600: error = 0.9984865, gradient norm = 0.0001015 (50 iteratio
ns in 5.186s)
[t-SNE] Iteration 650: error = 0.9830498, gradient norm = 0.0000958 (50 iteratio
ns in 5.202s)
[t-SNE] Iteration 700: error = 0.9726909, gradient norm = 0.0000877 (50 iteratio
ns in 5.171s)
[t-SNE] Iteration 750: error = 0.9647216, gradient norm = 0.0000823 (50 iteratio
ns in 5.171s)
[t-SNE] Iteration 800: error = 0.9582971, gradient norm = 0.0000755 (50 iteratio
ns in 5.186s)
[t-SNE] Iteration 850: error = 0.9531373, gradient norm = 0.0000697 (50 iteratio
ns in 5.155s)
[t-SNE] Iteration 900: error = 0.9484153, gradient norm = 0.0000696 (50 iteratio
ns in 5.186s)
[t-SNE] Iteration 950: error = 0.9445393, gradient norm = 0.0000659 (50 iteratio
ns in 5.186s)
[t-SNE] Iteration 1000: error = 0.9412127, gradient norm = 0.0000674 (50 iterati
ons in 5.186s)
[t-SNE] Error after 1000 iterations: 0.941213
```

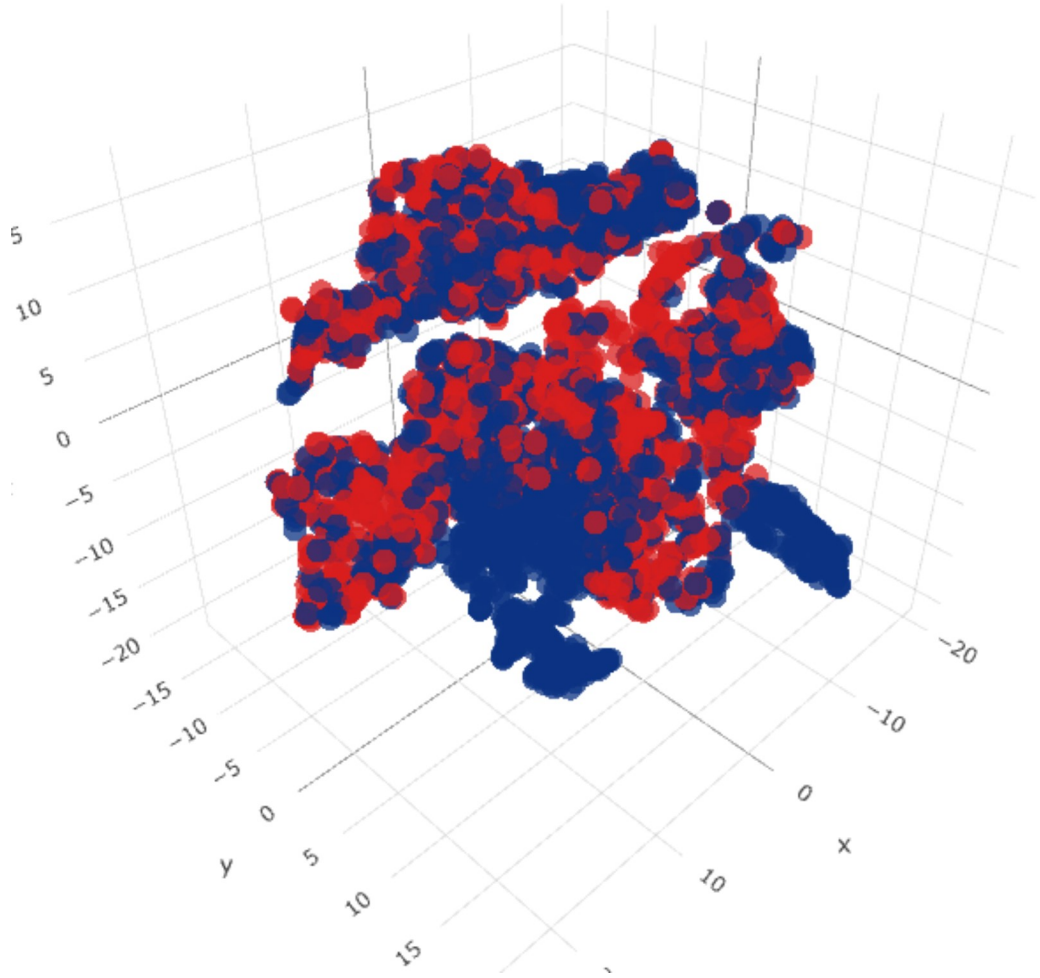
In [36]:
```
df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] ,'label':y})

# 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"
,markers=['s','o'])
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()
```



In [37]:
```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
     perplexity=30,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.016s...
[t-SNE] Computed neighbors for 5000 samples in 0.375s...
[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 0.203s
[t-SNE] Iteration 50: error = 80.3592682, gradient norm = 0.0335202 (50 iteratio
ns in 12.232s)
[t-SNE] Iteration 100: error = 69.1112671, gradient norm = 0.0036575 (50 iterati
ons in 6.389s)
[t-SNE] Iteration 150: error = 67.6171112, gradient norm = 0.0017708 (50 iterati
ons in 5.811s)
[t-SNE] Iteration 200: error = 67.0565109, gradient norm = 0.0011567 (50 iterati
ons in 5.702s)
[t-SNE] Iteration 250: error = 66.7296524, gradient norm = 0.0009161 (50 iterati
ons in 5.655s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 66.729652
[t-SNE] Iteration 300: error = 1.4983541, gradient norm = 0.0006807 (50 iteratio
ns in 6.873s)
[t-SNE] Iteration 350: error = 1.1549147, gradient norm = 0.0001922 (50 iteratio
ns in 8.592s)
[t-SNE] Iteration 400: error = 1.0101781, gradient norm = 0.0000912 (50 iteratio
ns in 8.670s)
[t-SNE] Iteration 450: error = 0.9388669, gradient norm = 0.0000628 (50 iteratio
ns in 8.576s)
[t-SNE] Iteration 500: error = 0.9029322, gradient norm = 0.0000524 (50 iteratio
ns in 8.482s)
[t-SNE] Iteration 550: error = 0.8841860, gradient norm = 0.0000482 (50 iteratio
ns in 8.357s)
[t-SNE] Iteration 600: error = 0.8722453, gradient norm = 0.0000365 (50 iteratio
ns in 8.311s)
[t-SNE] Iteration 650: error = 0.8627461, gradient norm = 0.0000347 (50 iteratio
ns in 8.123s)
[t-SNE] Iteration 700: error = 0.8549610, gradient norm = 0.0000312 (50 iteratio
ns in 8.139s)
[t-SNE] Iteration 750: error = 0.8487639, gradient norm = 0.0000311 (50 iteratio
ns in 8.123s)
[t-SNE] Iteration 800: error = 0.8440317, gradient norm = 0.0000281 (50 iteratio
ns in 8.154s)
[t-SNE] Iteration 850: error = 0.8396705, gradient norm = 0.0000250 (50 iteratio
ns in 8.154s)
[t-SNE] Iteration 900: error = 0.8354425, gradient norm = 0.0000242 (50 iteratio
ns in 8.123s)
[t-SNE] Iteration 950: error = 0.8317489, gradient norm = 0.0000233 (50 iteratio
ns in 8.092s)
[t-SNE] Iteration 1000: error = 0.8288577, gradient norm = 0.0000257 (50 iterati
ons in 8.061s)
[t-SNE] Error after 1000 iterations: 0.828858
```

In [38]:
```
trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

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


3d embedding with engineered features
5
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
y
5
10
15
−20
−10
0
x
10