KNN on Amzon Fine Food Review

Amazon Fine Food Review is about the reviews of customers on the food.

Number of columns: 10 Number of rows: 568454 Number of reviews: 568454 Number of products: 74258 Number of users: 256059

Attribute information:

1) Id: Number of rows

2) ProductId: Unique ID of product

3) UserId: User identification number

4) ProfileName: User name

5) HelpfulnessNumerator: Number of user found the review helpful

6) HelpfulnessDenominator: Number of user who found the review helpful or not

7) Score: Rating given to the product

8) Time: Timestamp at the time of review posted

9) Summary: Short version of text review

10) Text: Detailed text review

Objective:

We need to find if the review is positive (1) or negative (0).

We are provided with the score from 1 to 5. Let's assume score 1 & 2 are negative and score 4 & 5 are positive. We are ignoring 3 as it can be considered as neutral.

Loading the dataset

In [1]:
```
# Importing libraries

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# warnings library is to ignore warnings.
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Importing dataset

df = pd.read_csv('Reviews.csv')

# Displaying first 5 rows
df.head()
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5

Dataset information

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568454 entries, 0 to 568453
Data columns (total 10 columns):
Id                        568454 non-null int64
ProductId                 568454 non-null object
UserId                    568454 non-null object
ProfileName               568438 non-null object
HelpfulnessNumerator      568454 non-null int64
HelpfulnessDenominator    568454 non-null int64
Score                     568454 non-null int64
Time                      568454 non-null int64
Summary                   568427 non-null object
Text                      568454 non-null object
dtypes: int64(5), object(5)
memory usage: 43.4+ MB
```

In [4]:
```
print('Number of columns:' + ' ' + str(len(df.columns)))
print('Number of rows:' + ' ' + str(df['Id'].nunique()))
print('Number of reviews:' + ' ' + str(df['Id'].nunique()))
print('Number of products:' + ' ' + str(df['ProductId'].nunique()))
print('Number of users:' + ' ' + str(df['UserId'].nunique()))
```

```
Number of columns: 10
Number of rows: 568454
Number of reviews: 568454
Number of products: 74258
Number of users: 256059
```

Assign Polarity

Let us assign positive (1) to the score 4 and 5

Let us assign negative (0) to the score 1 and 2

Let us ignore score having 3

In [5]:
```
# Let us first create a new dataset which doesn't have score 3 and it's respective
rows.

print('Number of rows before removing score 3:' + ' ' + str(df['Id'].nunique()))

df_score = df[df['Score'] != 3]

print('Number of rows after removing score 3:' + ' ' + str(df_score['Id'].nunique()
))
```

```
Number of rows before removing score 3: 568454
Number of rows after removing score 3: 525814
```

In [6]:
```
# Defining a function to assign polarity.

def scr(sc):
    if sc > 3:
        return 1
    return 0
```

In [7]:
```
# Calling function to assign polarity using .apply()
df_score['Score'] = df_score['Score'].apply(scr)

print("Number of positive (1) and negative (0) reviews")

df_score['Score'].value_counts()
```

```
Number of positive (1) and negative (0) reviews
```

Out[7]:
```
1    443777
0     82037
Name: Score, dtype: int64
```

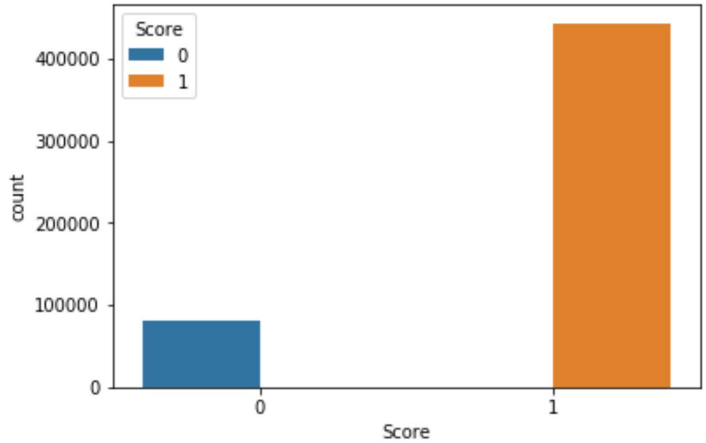
In [8]:
```
# Check the reduction percentage of data
print("Data percentage reduced to:" + ' ' + str(100*(len(df_score['Id'])/len(df['Id
']))))
```

```
Data percentage reduced to: 92.4989533014105
```

Exploratory Data Analysis

In [9]:
```
sns.countplot(df_score['Score'], hue = df_score['Score'])
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1c9f9358>



Observation:

As we can see, number of negative reviews are closer to 10k while positive review is more than 400k.

Ratio of negative and positive review can be assumed to be ~ 1:4.

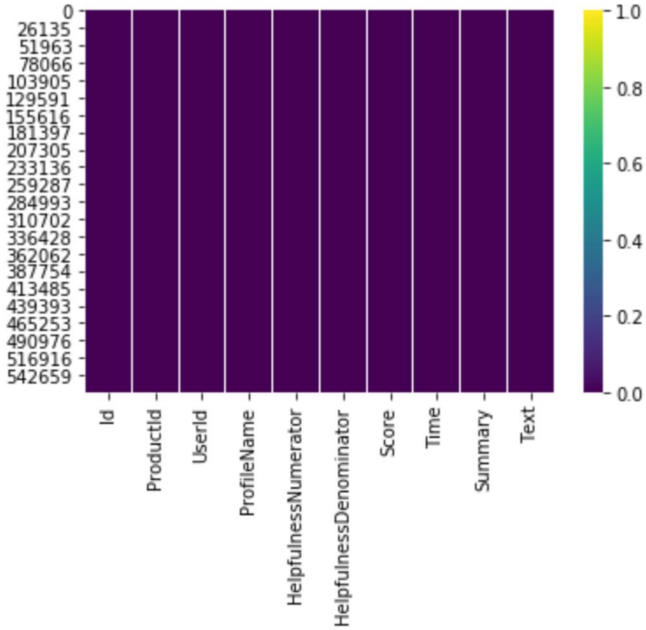
Also, we can conclude that ~90% of the reviews are positive.

Data Cleaning

Null values

In [10]:
```
sns.heatmap(df_score.isnull(), cmap = 'viridis')
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1cabc358>



Observation:

From the heat, it is clear that there are no null values

Duplication

In [11]:
```
# Check if there are any duplicates or not

df_score[(df_score['UserId'].duplicated() == True) & (df_score['ProfileName'].dupli
cated() == True) &
         (df_score['Time'].duplicated() == True) & (df_score['Text'].duplicated() =
= True)]
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenomina
29	30	B0001PB9FY	A3HDKO7OW0QNK4	Canadian Fan	1	
574	575	B000G6RYNE	A3PJZ8TU8FDQ1K	Jared Castle	2	
2309	2310	B0001VWE0M	AQM74O8Z4FMS0	Sunshine	0	
2323	2324	B0001VWE0C	AQM74O8Z4FMS0	Sunshine	0	
2336	2337	B0001FQVCK	A5D06XJHDXK75	C. Po	1	
2647	2648	B0016FY6H6	A2NLZ3M0OJV9NX	Mark Bodzin	0	
2653	2654	B0016FY6H6	A3I4PCBRENJNG2	L. Cain	0	
2946	2947	B0002TJAZK	A2ISKAWUPGGOLZ	M. S. Handley	0	
2947	2948	B0002TJAZK	A3TVZM3ZIXG8YW	christopher hayes	0	
3885	3886	B005GX7GVW	AS1FCKNKY95ID	Juli A. Lee "JingleJL"	1	
3886	3887	B005GX7GVW	A1I34N9LFOSCX7	Smeggy	0	
4640	4641	B0002NYO9I	A5DVX3B075B09	Patricia Kays	0	
4641	4642	B0002NYO9I	A376TWN7I4HMZ8	helios	0	
5397	5398	B00622CYVS	ATIHDHZYNQ0EI	Kristen O'donnell "twinsmom"	1	
5451	5452	B00622CYVI	ATIHDHZYNQ0EI	Kristen O'donnell "twinsmom"	2	

Observation:

We can see lot many duplicates in the dataset. Let's remove duplicates from the dataset. We should first sort the dataset by ProductId so that we can keep the 1st review and remove duplicates without any mess.

In [12]:
```
# First let's sort the dataset by product ids.
df_sorted = df_score.sort_values('ProductId', axis = 0, ascending = True)

# Now removing duplicates from the sorted dataset.
df_dup = df_sorted.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, k
eep='first', inplace=False)

print("Data percentage reduced to after removing duplicates:" + ' ' + str(100*(len(
df_dup['Id'])/len(df['Id']))))
```

```
Data percentage reduced to after removing duplicates: 64.06375889693801
```

Compare HelpfulnessNumerator and HelpfulnessDenominator

HelpfulnessNumerator should always be less than or equal to HelpfulnessDenominator

In [13]:
```
df_dup[df_dup['HelpfulnessNumerator'] > df_dup['HelpfulnessDenominator']]
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	S
64421	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	
44736	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	

Observation:

We can see 2 data where HelpfulnessNumerator is greater than HelpfulnessDenominator. We need to remove such data.

In [14]:
```
df_final = df_dup[df_dup['HelpfulnessNumerator'] <= df_dup['HelpfulnessDenominator'
]]

# Check if there is any data where HelpfulnessNumerator is greater than Helpfulness
Denominator
df_final[df_final['HelpfulnessNumerator'] > df_final['HelpfulnessDenominator']]
```

Out[14]:

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary

We can see now that the dataset has HelpfulnessNumerator less than or equal HelpfulnessDenominator.

In [15]:
```
print("Data percentage reduced to:" + ' ' + str(100*(len(df_final['Id'])/len(df['Id
']))))
```

```
Data percentage reduced to: 64.06340706547935
```

Text Preprocessing:

1) Remove HTML tag

2) Remove punctuations and numbers

3) Remove URLs

4) Renaming short forms like can't to can not, 's to is, 're to are, etc

5) Stop-word removal

6) Stemming

7) Change it to lowercase and join it.

In [16]:
```
# Importing libraries

import re
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from bs4 import BeautifulSoup
from tqdm import tqdm

# Create an instance for SnowballStemmer
ss = SnowballStemmer('english')
```

In [17]:
```
# Let's create a sample text to see how text cleaning works.
sam = "I don't know if it's the cactus or the tequila or just the unique combinatio
n of ingredients,. But I went through https://www.amazon.in/. but the flavour of th
is hot sauce makes it one of a kind! When we realized that we simply couldn't find
it anywhere in our city we were bummed.<br /><br />Now, because of the magic of the
internet, we have a case of the sauce and are ecstatic because of it.<br /><br />If
you love hot sauce..I mean really love hot sauce, but don't want a sauce that taste
lessly burns your throat, grab a bottle of Tequila Picante Gourmet de Inclan.  Just
realize that once you taste it, you will never want to use any other sauce.<br /><b
r />Thank you for the personal, incredible service!"
sam
```

Out[17]:
```
"I don't know if it's the cactus or the tequila or just the unique combination o
f ingredients,. But I went through https://www.amazon.in/. but the flavour of th
is hot sauce makes it one of a kind! When we realized that we simply couldn't fi
nd it anywhere in our city we were bummed.<br /><br />Now, because of the magic
of the internet, we have a case of the sauce and are ecstatic because of it.<br
/><br />If you love hot sauce..I mean really love hot sauce, but don't want a sa
uce that tastelessly burns your throat, grab a bottle of Tequila Picante Gourmet
de Inclan.  Just realize that once you taste it, you will never want to use any
other sauce.<br /><br />Thank you for the personal, incredible service!"
```

Observation:

1) HTML tags: Text contains html tags like '< br />' '< br />', etc...

2) URL: Text contains URL like https://www.amazon.in/ (https://www.amazon.in/)

3) Short words: Text contains short words like couldn't, it's, etc...

4) Punctuation: Text contains punctuations like !.,"?"":

HTML tag

Defining function to remove HTML tag,

In [18]:
```
# To remove HTML tags
def html(ht):
    ht = BeautifulSoup(ht, 'lxml').get_text()
    return ht
```

In [19]:
```
# Check html removal funcation on sample text sam

print("Text before removing HTML tag:",'\n')
print(sam, '\n')
print('*'*50, '\n')

print("Text after removing HTML tag:")
html(sam)
```

```
Text before removing HTML tag:

I don't know if it's the cactus or the tequila or just the unique combination of
ingredients,. But I went through https://www.amazon.in/. but the flavour of this
hot sauce makes it one of a kind! When we realized that we simply couldn't find
it anywhere in our city we were bummed.<br /><br />Now, because of the magic of
the internet, we have a case of the sauce and are ecstatic because of it.<br /><
br />If you love hot sauce..I mean really love hot sauce, but don't want a sauce
that tastelessly burns your throat, grab a bottle of Tequila Picante Gourmet de
Inclan.  Just realize that once you taste it, you will never want to use any oth
er sauce.<br /><br />Thank you for the personal, incredible service!

**************************************************

Text after removing HTML tag:
```

Out[19]:
```
"I don't know if it's the cactus or the tequila or just the unique combination o
f ingredients,. But I went through https://www.amazon.in/. but the flavour of th
is hot sauce makes it one of a kind! When we realized that we simply couldn't fi
nd it anywhere in our city we were bummed.Now, because of the magic of the inter
net, we have a case of the sauce and are ecstatic because of it.If you love hot
sauce..I mean really love hot sauce, but don't want a sauce that tastelessly bur
ns your throat, grab a bottle of Tequila Picante Gourmet de Inclan.  Just realiz
e that once you taste it, you will never want to use any other sauce.Thank you f
or the personal, incredible service!"
```

All the html tags have been removed

URL removal

Defining a funtion to remove URLs

In [20]:
```
def url(ur):
    ur = re.sub(r"http\S+", '', ur)
    return ur
```

In [21]:
```
# Check URL removal funcation on sample text sam

print("Text before removing URL:",'\n')
print(sam, '\n')
print('*'*50, '\n')

print("Text after removing URL:")
url(sam)
```

```
Text before removing URL:

I don't know if it's the cactus or the tequila or just the unique combination of
ingredients,. But I went through https://www.amazon.in/. but the flavour of this
hot sauce makes it one of a kind! When we realized that we simply couldn't find
it anywhere in our city we were bummed.<br /><br />Now, because of the magic of
the internet, we have a case of the sauce and are ecstatic because of it.<br /><
br />If you love hot sauce..I mean really love hot sauce, but don't want a sauce
that tastelessly burns your throat, grab a bottle of Tequila Picante Gourmet de
Inclan.  Just realize that once you taste it, you will never want to use any oth
er sauce.<br /><br />Thank you for the personal, incredible service!

**************************************************

Text after removing URL:
```

Out[21]:
```
"I don't know if it's the cactus or the tequila or just the unique combination o
f ingredients,. But I went through  but the flavour of this hot sauce makes it o
ne of a kind! When we realized that we simply couldn't find it anywhere in our c
ity we were bummed.<br /><br />Now, because of the magic of the internet, we hav
e a case of the sauce and are ecstatic because of it.<br /><br />If you love hot
sauce..I mean really love hot sauce, but don't want a sauce that tastelessly bur
ns your throat, grab a bottle of Tequila Picante Gourmet de Inclan.  Just realiz
e that once you taste it, you will never want to use any other sauce.<br /><br /
>Thank you for the personal, incredible service!"
```

URL have been removed.

Short word to full word

Defining a function to convert short words like couldn't to full word could not

In [22]:
```
def short_word(full_word):

    full_word = full_word.lower()           # Python reads Won't and won't as se
parate words. So change to lowercase

    full_word = re.sub(r"won't", "will not", full_word)
    full_word = re.sub(r"wouldn't", "would not", full_word)
    full_word = re.sub(r"can't", "can not", full_word)
    full_word = re.sub(r"don't", "don not", full_word)
    full_word = re.sub(r"shouldn't", "should not", full_word)
    full_word = re.sub(r"couldn't", "could not", full_word)
    full_word = re.sub(r"\'re", " are", full_word)
    full_word = re.sub(r"\'s", " is", full_word)
    full_word = re.sub(r"\'d", " would", full_word)
    full_word = re.sub(r"\'ll", " will", full_word)
    full_word = re.sub(r"\'ve", " have", full_word)
    full_word = re.sub(r"\'m", " am", full_word)

    return full_word
```

In [23]:
```
# Check coversion of words from short form to full form

print("Text before converting from short word to full word:",'\n')
print(sam, '\n')
print('*'*50, '\n')

print("Text after converting from short word to full word:")
short_word(sam)
```

```
Text before converting from short word to full word:

I don't know if it's the cactus or the tequila or just the unique combination of
ingredients,. But I went through https://www.amazon.in/. but the flavour of this
hot sauce makes it one of a kind! When we realized that we simply couldn't find
it anywhere in our city we were bummed.<br /><br />Now, because of the magic of
the internet, we have a case of the sauce and are ecstatic because of it.<br /><
br />If you love hot sauce..I mean really love hot sauce, but don't want a sauce
that tastelessly burns your throat, grab a bottle of Tequila Picante Gourmet de
Inclan.  Just realize that once you taste it, you will never want to use any oth
er sauce.<br /><br />Thank you for the personal, incredible service!

**************************************************

Text after converting from short word to full word:
```

Out[23]:
```
'i don not know if it is the cactus or the tequila or just the unique combinatio
n of ingredients,. but i went through https://www.amazon.in/. but the flavour of
this hot sauce makes it one of a kind! when we realized that we simply could not
find it anywhere in our city we were bummed.<br /><br />now, because of the magi
c of the internet, we have a case of the sauce and are ecstatic because of it.<b
r /><br />if you love hot sauce..i mean really love hot sauce, but don not want
a sauce that tastelessly burns your throat, grab a bottle of tequila picante gou
rmet de inclan.  just realize that once you taste it, you will never want to use
any other sauce.<br /><br />thank you for the personal, incredible service!'
```

Punctuation and numeric removal and snoballstemming

Defining a function to remove punctuations and numerics and also stemming.

In [24]:
```
def punc(pun):

    pun = re.sub('[^a-zA-Z]', ' ', pun)
    pun = pun.lower()
    pun = pun.split()

    if len(pun) > 2:
        pun = [ss.stem(sw) for sw in pun if sw not in stopwords.words('english')]
        pun = ' '.join(pun)
        return pun
```

Creating new dataframe with few datapoints

In [25]:
```
# Creating new dataframe with 8k points

df_pos_4k = df_final[df_final['Score']==1].sample(n = 4000)
df_neg_4k = df_final[df_final['Score']==0].sample(n = 4000)

# Concat df_pos and df_neg
df_8k = pd.concat([df_pos_4k, df_neg_4k])
```

In [26]:
```
print("Shape of df_8k:" + ' ' + str(df_8k.shape))
print("Number of positive review in df_8k:" + ' ' + str(len(df_8k[df_8k['Score'] ==
1])))
print("Number of negative review in df_8k:" + ' ' + str(len(df_8k[df_8k['Score'] ==
0])))
```

```
Shape of df_8k: (8000, 10)
Number of positive review in df_8k: 4000
Number of negative review in df_8k: 4000
```

Apply text processing functions to 'Text' data

HTML tag removal, url, punctuation, stop word removal, rename

In [27]:
```
filtered = []                                          # All the filtered data is stored
in this list
positive = []                                          # All the positive review data is
stored in this list
negative = []                                          # All the negative review data is
stored in this list

for i, s in enumerate (tqdm(df_8k['Text'].values)):

    h = html(s)                                               # Removes HTML tag
    u = url(h)                                                # Removes URL
    f = short_word(u)                                         # Converts from short form
to full form
    p = punc(f)                                               # Removes punctuation, numb
ers, does stemming for the words > 2

    if df_8k['Score'].values[i] == 1:
        positive.append(p)                                     # Positive review list
    if df_8k['Score'].values[i] == 0:
        negative.append(p)                                     # Negative review list

    filtered.append(p)                                         # Complete filtered list
```

```
100%|██████████████████████████████████████| 8000/8000 [13:05<00:00, 10.18it/s]
```

Apply text processing functions to 'Summary' data

HTML tag removal, url, punctuation, stop word removal, rename

In [28]:
```
filtered_smr = []                                          # All the filtered data is sto
red in this list
positive_smr = []                                          # All the positive review data
is stored in this list
negative_smr = []                                          # All the negative review data
is stored in this list

for i, s in enumerate (tqdm(df_8k['Summary'].values)):

    h = html(s)                                               # Removes HTML tag
    u = url(h)                                                # Removes URL
    f = short_word(u)                                         # Converts from short form
to full form
    p = punc(f)                                               # Removes punctuation, numb
ers, does stemming for the words > 2

    if df_8k['Score'].values[i] == 1:
        positive_smr.append(p)                                     # Positive review list
    if df_8k['Score'].values[i] == 0:
        negative_smr.append(p)                                     # Negative review list

    filtered_smr.append(p)                                         # Complete filtered li
st
```

```
100%|█████████████████████████████████████| 8000/8000 [00:48<00:00, 165.28it/s]
```

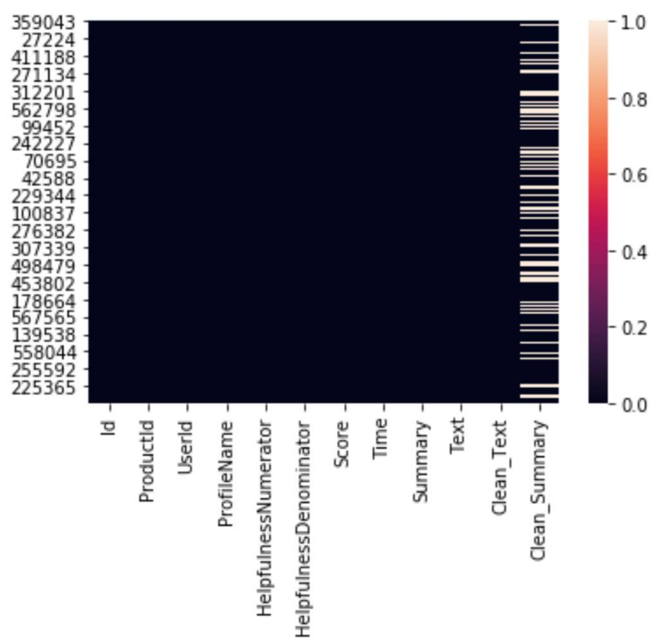
In [29]:
```
# Adding a new column to df_8k which contains clean text from Text column
df_8k['Clean_Text'] = filtered

# Adding filtered summary data to df_1k
df_8k['Clean_Summary'] = filtered_smr

# 2 new columns df_1k['clean_Text'] and df_1k['Clean_Summary'] have been added.
df_8k.head(2)
```

Out[29]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominato
359043	359044	B006MONQDQ	A2C7R9GGHY107Z	Ms. Meticulous	1	:
555390	555391	B005P0WL1G	ABRVCFPVBT8VH	Amanda R. Valentine	5	!

In [30]:
```
# After cleaning text data, let's check if there is any null values

sns.heatmap(df_8k.isnull())
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x43f62d68>



Observation:

After cleaning both Text and Summary data, we can see lots of null values in the Clean_Summary column. So, working on Clean_Summary column is not worthy.

Time based splitting

We need to sort the data based on time in an increasing order.

Reason: We need to predict the new input data based on the built KNN model. Built model should be trained based on increasing time order which is more helpfull in building a better model to predict new input data.

In [31]:
```
df_8k = df_8k.sort_values('Time', ascending = True)
```

In [32]:
```
# Saving the file which is sorted based on time

df_8k.to_csv('Amazon_8k_time.csv', index_label = False)
```

Frequency Distribution of words

In [33]:
```
# Import nltk
import nltk

freq_pos = nltk.FreqDist(positive)
freq_neg = nltk.FreqDist(negative)

print("Frequency distribution of positive review:" + ' ' + str(freq_pos.most_common
(1)), '\n')
print('*'*50, '\n')
print("Frequency distribution of negative review:" + ' ' + str(freq_neg.most_common
(1)))
```

```
Frequency distribution of positive review: [('great new product tri flavor tropi
c citrus fruit punch acai grape pomegran energi peach green tea work high school
alway get sick drink vitamin squeez sever month sick thing done differ like tast
tap water school product ideal chang tap water drink delici motiv drink water fi
nal truli great drink much better mio half price vitaminwat take littl squeez vi
tamin squeez leav aftertast like calori product calori carb sugar love teen thin
k tropic citrus tast like tang creamsicl head back safeway get high recommend pr
oduct', 1)]

**************************************************

Frequency distribution of negative review: [('take reason size pot plant take tw
o liter spring water pour said water pot plant wait short drain liquid emerg bot
tom pot pour liquid clear plastic bottl write blk place bottl onto shelf post pr
ice tag half quart spring water infus fulvic acid lol sucker born everi minut yu
p right', 2)]
```

Observation:

As we can see, in the negative review there is word 'like' which actually is positive but here it is considered as negative because 'not' word has been removed by the stop word feature. We need to avoid such word removal. It is beter to use n-gram preferebally bi-gram.

Train Test Split

Splitting the data in to train and test split

Note: Creating 2 train and 2 test split for 8k and 1k dataset each.

In [34]:
```
x_8k = df_8k['Clean_Text']
print("Shape of x_8k is:" + ' ' + str(x_8k.shape))

y_8k = df_8k['Score']
print("Shape of y_8k is:" + ' ' + str(y_8k.shape),'\n')
```

```
Shape of x_8k is: (8000,)
Shape of y_8k is: (8000,)
```

In [35]:
```
# Import train_test_split library
from sklearn.model_selection import train_test_split

# Split 8k data into train and test set
x_train_8k, x_test_8k, y_train_8k, y_test_8k = train_test_split(x_8k, y_8k, test_si
ze = 0.2,
                                                                 shuffle =  False, r
andom_state = 0)
```

KNN

There are 2 methods which can be used to find whether or not review is positive.

Method-1: K-fold cross validation

Method-2: Simple cross validation

K-fold cross validation - Bag of Words

In [36]:
```
# Import CountVectorizer library
from sklearn.feature_extraction.text import CountVectorizer

# Create an instance for CountVectorizer.
# Bi-gram vector
cv = CountVectorizer(ngram_range = (1,2))

# Fit and transform the x_train
x_train_8k_bow = cv.fit_transform(x_train_8k)

# Transform the x_test
x_test_8k_bow = cv.transform(x_test_8k)

print("Type of x_train_8k_bow:" + ' ' + str(type(x_train_8k_bow)))
print("Shape of x_train_8k_bow:" + ' ' + str(x_train_8k_bow.get_shape()))
print("Number of unique words in x_train_8k_bow:" + ' ' + str(x_train_8k_bow.get_sh
ape()[1]))


# Normalizing the data to get everything in a single scale
# Import normalization library

from sklearn.preprocessing import normalize

x_train_8k_bow = normalize(x_train_8k_bow)
x_test_8k_bow = normalize(x_test_8k_bow)
```

```
Type of x_train_8k_bow: <class 'scipy.sparse.csr.csr_matrix'>
Shape of x_train_8k_bow: (6400, 172830)
Number of unique words in x_train_8k_bow: 172830
```

TimeSeriesSplit

Since the data is time series based, we need to apply time series split such that train data is further divided into 2 parts i.e training and cross validation data.

This train and cv data is used to build a model and predict in order to get the best k value based on accuracy. Then using this K value we can get close to the previously predicted accuracy for the new data which in our case is test data.

In [37]:
```
# Import TimeSeriesSplit library
from sklearn.model_selection import TimeSeriesSplit

# Create an object
ts = TimeSeriesSplit(n_splits = 10)

for train, cv in ts.split(x_train_8k_bow):
    print(x_train_8k_bow[train].shape, x_train_8k_bow[cv].shape )
```

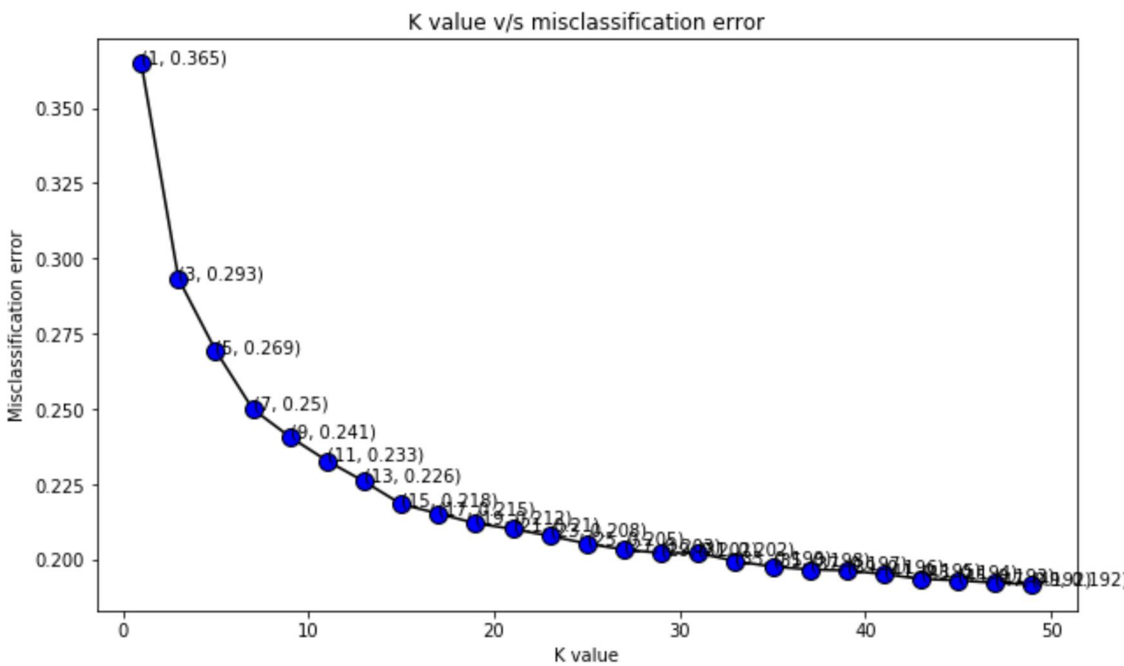
```
(590, 172830) (581, 172830)
(1171, 172830) (581, 172830)
(1752, 172830) (581, 172830)
(2333, 172830) (581, 172830)
(2914, 172830) (581, 172830)
(3495, 172830) (581, 172830)
(4076, 172830) (581, 172830)
(4657, 172830) (581, 172830)
(5238, 172830) (581, 172830)
(5819, 172830) (581, 172830)
```

Brute Force with 10 fold cross validation

In [38]:
```
# Import required libraries

import scikitplot as skplt
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

In [39]:
```
# Input the range of odd numbers

num = range(1,50,2)

cv_score = []              # It stores the cross_val_score

time = TimeSeriesSplit(n_splits=10)

for k in tqdm(num):
    knn = KNeighborsClassifier(n_neighbors = k, algorithm = 'brute', n_jobs = -1)
    score = cross_val_score(knn, x_train_8k_bow, y_train_8k, cv = time, scoring = '
roc_auc')
    cv_score.append(score.mean())

# Misclassification

print("Misclassification errors are:", '\n')
mse = [1-x for x in cv_score]
print(mse, '\n')
print('*'*100)

optimal_k_val = num[mse.index(min(mse))]
print("The best K value is:" + ' ' + str(optimal_k_val), '\n')
print('*'*100)

# To find the best K value, we can analyze from the plot (num vs misclassification
error)

plt.figure(figsize = (10,6))
plt.plot(num, mse, color = 'black', marker = 'o', markerfacecolor = 'blue', markers
ize = 10)

plt.title("K value v/s misclassification error")
plt.xlabel("K value")
plt.ylabel("Misclassification error")

# Adding annotation to get the marker position number
for xy in zip(num, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)
```

```
100%|██████████████████████████████████████████| 25/25 [01:29<00:00,  3.96s/it]

Misclassification errors are:

[0.3645954602839758, 0.2932196489412021, 0.269395941788635, 0.24997564657768456,
0.24076599678467792, 0.23281546562060018, 0.225950890048512 54, 0.218427370549087
78, 0.21529519461852964, 0.21205135709309553, 0.21017563710547038, 0.20789712282
75346, 0.20532670300833178, 0.2033208009492974, 0.20222122299312584, 0.202190459
84678174, 0.1994914216555379, 0.1976780802840652, 0.19672514709995303, 0.1964248
651890257, 0.19518127212804592, 0.1935838489142533, 0.19311116958870078, 0.19230
970278829496, 0.19179103460282543]

****************************************************************************************************
The best K value is: 49

****************************************************************************************************
```



Prediction

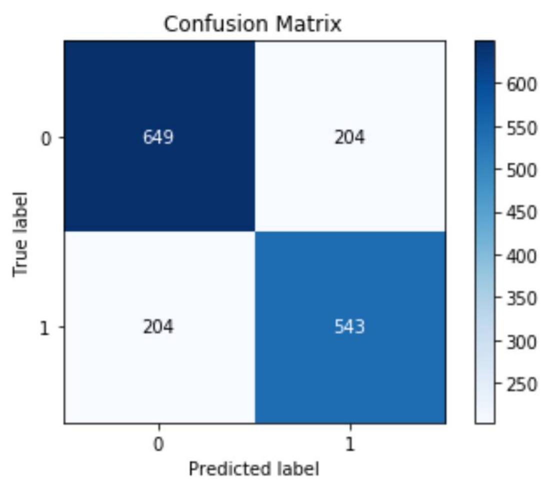
As we see, best or optimal k-value we obtained for the respective error is low is 49.

We will now use the k value 49 for the prediction

In [40]:
```
knn = KNeighborsClassifier(n_neighbors = optimal_k_val)
knn_fit = knn.fit(x_train_8k_bow, y_train_8k)
prediction = knn_fit.predict(x_test_8k_bow)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_8k, prediction)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_8k, prediction))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_8k, prediction))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_8k, prediction)
```

```
Accuracy score

74.5

********************************************************************************
********************
Classification report

              precision    recall  f1-score   support

           0       0.76      0.76      0.76       853
           1       0.73      0.73      0.73       747

   micro avg       0.74      0.74      0.74      1600
   macro avg       0.74      0.74      0.74      1600
weighted avg       0.74      0.74      0.74      1600

********************************************************************************
********************
Confusion matrix
[[649 204]
 [204 543]]
Confusion matrix table:
```

Out[40]: `<matplotlib.axes._subplots.AxesSubplot at 0x55f72cc0>`



Tf-Idf

In [41]:
```
# Import Tf-Idf
from sklearn.feature_extraction.text import TfidfVectorizer

# Create an instance
tf = TfidfVectorizer(ngram_range = (1,2))

# Fit and transform the x_train
x_train_8k_tf = tf.fit_transform(x_train_8k)

# Transform the x_test
x_test_8k_tf = tf.transform(x_test_8k)

print("Type of x_train_8k_tf:" + ' ' + str(type(x_train_8k_tf)))
print("Shape of x_train_8k_tf:" + ' ' + str(x_train_8k_tf.get_shape()))
print("Number of unique words in x_train_8k_tf:" + ' ' + str(x_train_8k_tf.get_shap
e()[1]))


# Normalize train and test data
x_train_8k_tf = normalize(x_train_8k_tf)
x_test_8k_tf = normalize(x_test_8k_tf)
```

```
Type of x_train_8k_tf: <class 'scipy.sparse.csr.csr_matrix'>
Shape of x_train_8k_tf: (6400, 172830)
Number of unique words in x_train_8k_tf: 172830
```

TimeSeriesSplit

Since the data is time series based, we need to apply time series split such that train data is further divided into 2 parts i.e training and cross validation data.

This train and cv data is used to build a model and predict in order to get the best k value based on accuracy. Then using this K value we can get close to the previously predicted accuracy for the new data which in our case is test data.

In [42]:
```
# Import TimeSeriesSplit library
from sklearn.model_selection import TimeSeriesSplit

# Create an object
ts = TimeSeriesSplit(n_splits = 10)

for train_8k_tf, cv_8k_tf in ts.split(x_train_8k_tf):
    print(x_train_8k_tf[train_8k_tf].shape, x_train_8k_tf[cv_8k_tf].shape )
```

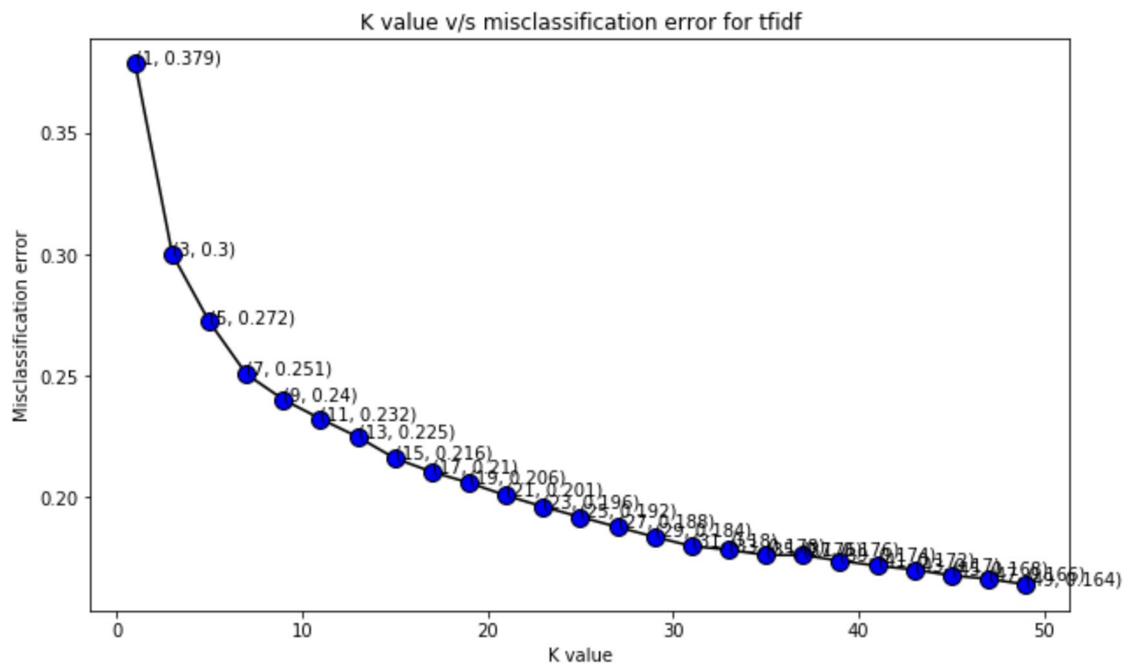
```
(590, 172830) (581, 172830)
(1171, 172830) (581, 172830)
(1752, 172830) (581, 172830)
(2333, 172830) (581, 172830)
(2914, 172830) (581, 172830)
(3495, 172830) (581, 172830)
(4076, 172830) (581, 172830)
(4657, 172830) (581, 172830)
(5238, 172830) (581, 172830)
(5819, 172830) (581, 172830)
```

Brute force with 10 fold cross validation

In [43]:

```
# Input the range of odd numbers

num = range(1,50,2)

cv_score_tf = []              # It stores the cross_val_score

time = TimeSeriesSplit(n_splits=10)

for k in tqdm(num):
    knn = KNeighborsClassifier(n_neighbors = k, algorithm = 'brute', n_jobs = -1)
    score_tf = cross_val_score(knn, x_train_8k_tf, y_train_8k, cv = time, scoring =
'roc_auc')
    cv_score_tf.append(score_tf.mean())

# Misclassification error

print("Misclassification errors are:", '\n')
mse_tf = [1-x for x in cv_score_tf]
print(mse_tf, '\n')
print('*'*100)

optimal_k_tf = num[mse_tf.index(min(mse_tf))]
print("The best K value is:" + ' ' + str(optimal_k_tf), '\n')
print('*'*100)

# To find the best K value, we can analyze from the plot (num vs misclassification
error)

plt.figure(figsize = (10,6))
plt.plot(num, mse_tf, color = 'black', marker = 'o', markerfacecolor = 'blue', mark
ersize = 10)

plt.title("K value v/s misclassification error for tfidf")
plt.xlabel("K value")
plt.ylabel("Misclassification error")

# Adding annotation to get the marker position number
for xy in zip(num, np.round(mse_tf,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)
```

```
100%|██████████████████████████████████████████| 25/25 [01:32<00:00,  3.50s/it]

Misclassification errors are:

[0.3785010968881559, 0.3001074699464743, 0.2721821913900164, 0.25059520822738535
, 0.24011439736746987, 0.23218219466866574, 0.22476152288809093, 0.2159617633877
724, 0.21032964294383283, 0.2058560225805639, 0.20058706517903901, 0.19597400946
50162, 0.19174636099718256, 0.18763581500451, 0.1835548278673217, 0.179748383445
42388, 0.17820079735630912, 0.1760575853673938, 0.1760112142265724, 0.1736867564
0005815, 0.17164858051561838, 0.17003937854506324, 0.16764345182538132, 0.166207
66711228807, 0.1639722452037864]

********************************************************************************
********************
The best K value is: 49

********************************************************************************
********************
```



Prediction

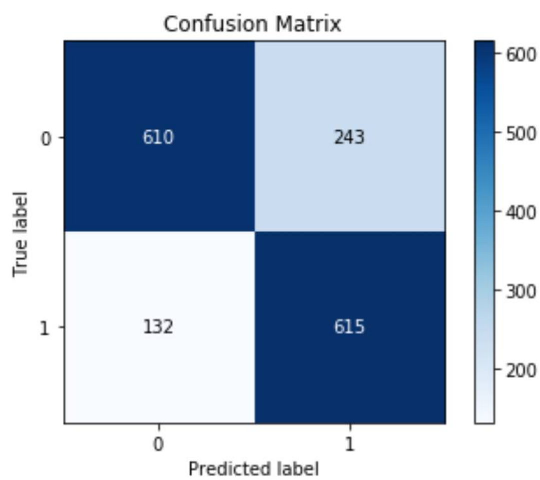
We got the best k value of 49 with lowest error

We will use this best k value 49 for the prediction.

In [44]:
```
knn_lib_tf = KNeighborsClassifier(n_neighbors = optimal_k_tf)
knn_fit_tf = knn_lib_tf.fit(x_train_8k_tf, y_train_8k)
prediction_tf = knn_fit_tf.predict(x_test_8k_tf)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_8k, prediction_tf)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_8k, prediction_tf))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_8k, prediction_tf))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_8k, prediction_tf)
```

```
Accuracy score

76.5625

********************************************************************************
********************
Classification report

              precision    recall  f1-score   support

           0       0.82      0.72      0.76       853
           1       0.72      0.82      0.77       747

   micro avg       0.77      0.77      0.77      1600
   macro avg       0.77      0.77      0.77      1600
weighted avg       0.77      0.77      0.77      1600

********************************************************************************
********************
Confusion matrix
[[610 243]
 [132 615]]
Confusion matrix table:
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x5631b128>



Word2Vec

In [45]:

```
# Training text corpus to build vocabulary

w2v = []
for w in tqdm(df_8k['Text'].values):

    hl = html(w)
    hl = url(hl)
    hl = short_word(hl)
    hl = re.sub('[^a-zA-Z]', ' ', hl)

    w2v.append(hl.split())
```

```
100%|████████████████████████████████████████| 8000/8000 [00:12<00:00, 615.97it/s]
```

In [46]:
```
# import gensim library
import gensim

#Create an instance for the genism model
w2v_model = gensim.models.Word2Vec(w2v, min_count = 5, size = 50, workers = 4)

print(w2v_model)


# Creating own corpus vocabulary
w2v_vocab = w2v_model[w2v_model.wv.vocab]

print("Shape of w2v_vocab:" + ' ' + str(w2v_vocab.shape))


# Creating list of words
w2v_word = list(w2v_model.wv.vocab)

print("Length of w2v_word:" + ' ' + str(len(w2v_word)), '\n')

print('*'*50)

print("First 10 words from the list of words w2v_word:")
w2v_word[:10]
```

```
Word2Vec(vocab=5873, size=50, alpha=0.025)
Shape of w2v_vocab: (5873, 50)
Length of w2v_word: 5873

**************************************************
First 10 words from the list of words w2v_word:
```

Out[46]:
```
['i',
 'was',
 'very',
 'skeptical',
 'when',
 'bought',
 'this',
 'item',
 'so',
 'imagine']
```

In [47]:
```
# Let's check the most similar words

w2v_model.wv.most_similar('wonder')
```

Out[47]:
```
[('dislike', 0.8707257509231567),
 ('mean', 0.8622766733169556),
 ('yeah', 0.8582401275634766),
 ('trust', 0.8550702333450317),
 ('disagree', 0.8520330786705017),
 ('edible', 0.8464459180831909),
 ('honestly', 0.8462368249893188),
 ('assume', 0.844734787940979),
 ('happens', 0.841597318649292),
 ('doubt', 0.8406253457069397)]
```

In [48]:
```
# Let's check the most similar words

w2v_model.wv.most_similar('buy')
```

Out[48]:
```
[('purchase', 0.8327258229255676),
 ('recommend', 0.8310237526893616),
 ('probably', 0.7980687022209167),
 ('buying', 0.7953695058822632),
 ('purchasing', 0.7622392773628235),
 ('consider', 0.7526281476020813),
 ('expect', 0.7382147908210754),
 ('believe', 0.7359397411346436),
 ('definitely', 0.7322167158126831),
 ('rate', 0.7293595671653748)]
```

Observation:

As we can see, .most_similar gives the similar words to the input word along with the percentage of similarity

Average Word2Vec

Convert Word2Vec to vectors.

Average Word2Vec is nothing but the average of vectors of each word of a given text/review/sentence.

In [49]:
```
sentence = [] # avg w2v of sentence/review will be stored in the empty list

for sen in tqdm(w2v):
    zero = np.zeros(50)  # (50,) matrix which is initial to add to the first w2v of
word in a sentence/review
    count_div = 0          # Increases by 1 every iteration and divides the w2v the
sum of w2v sentence/review
    for word in sen:
        if word in w2v_word:
            vec = w2v_model.wv[word]  # Gets the w2v for each word in a sentence/re
view
            zero += vec              # Sums the w2v of each word in a sentence/revi
ew at every iteration
            count_div += 1           # Increases by 1 at every iteration
    if count_div != 0:
        zero /= count_div            # w2v of sentence/review is divided by total n
umber of words in a sentence/review (average w2v)
    sentence.append(zero)            # Stores all the avg w2z in an empty list sent
ence
```

```
100%|████████████████████████████████████| 8000/8000 [01:00<00:00, 133.31it/s]
```

In [50]:
```
# Normalize the data

sentence_8k_avg_norm = normalize(sentence)
```

Train Test Split

In [51]:
```
#Splitting into train and test data.

x_train_8k_avg, x_test_8k_avg_t, y_train_8k, y_test_8k = train_test_split(sentence_
8k_avg_norm, y_8k, test_size = 0.2,
                                                             random_state = 0, shuf
fle = False )
```

Time Series Split

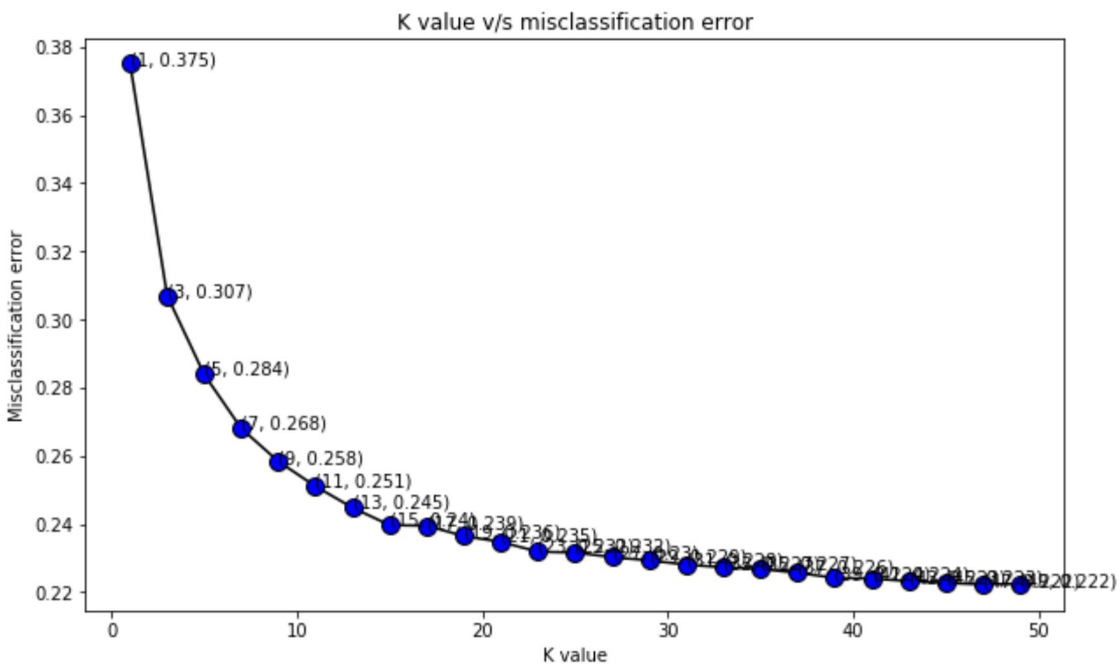
In [52]:
```
# Import TimeSeriesSplit library
from sklearn.model_selection import TimeSeriesSplit

# Create an object
ts = TimeSeriesSplit(n_splits = 10)

for train_8k_avg, cv_8k_avg in ts.split(x_train_8k_avg):
    print(x_train_8k_avg[train_8k_avg].shape, x_train_8k_avg[cv_8k_avg].shape )
```

```
(590, 50) (581, 50)
(1171, 50) (581, 50)
(1752, 50) (581, 50)
(2333, 50) (581, 50)
(2914, 50) (581, 50)
(3495, 50) (581, 50)
(4076, 50) (581, 50)
(4657, 50) (581, 50)
(5238, 50) (581, 50)
(5819, 50) (581, 50)
```

Brute force with 10 fold cross validation

In [53]:
```
# Input the range of odd numbers

num = range(1,50,2)

cv_score = []             # It stores the cross_val_score

time = TimeSeriesSplit(n_splits=10)

for k in tqdm(num):
    knn = KNeighborsClassifier(n_neighbors = k, algorithm = 'brute', n_jobs = -1)
    score = cross_val_score(knn, x_train_8k_avg, y_train_8k, cv = time, scoring = '
roc_auc')
    cv_score.append(score.mean())

# Misclassification

print("Misclassification errors are:", '\n')
mse = [1-x for x in cv_score]
print(mse, '\n')
print('*'*100)

optimal_k_val = num[mse.index(min(mse))]
print("The best K value is:" + ' ' + str(optimal_k_val), '\n')
print('*'*100)

# To find the best K value, we can analyze from the plot (num vs misclassification
error)

plt.figure(figsize = (10,6))
plt.plot(num, mse, color = 'black', marker = 'o', markerfacecolor = 'blue', markers
ize = 10)

plt.title("K value v/s misclassification error")
plt.xlabel("K value")
plt.ylabel("Misclassification error")

# Adding annotation to get the marker position number
for xy in zip(num, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)
```

```
100%|████████████████████████████████████████| 25/25 [00:36<00:00,  1.14s/it]

Misclassification errors are:

[0.3750178559380083, 0.30699527788981906, 0.2838369767808262, 0.2680050004373784
, 0.2583446331634176, 0.25104218414546553, 0.2447617555522954, 0.239656907743859
17, 0.23949088242021566, 0.23644592899671113, 0.23472228439325993, 0.23186565455
938513, 0.23159831377739093, 0.23026638782741438, 0.22929863269402717, 0.2280667
8613663478, 0.22726241093804944, 0.2267929092662193, 0.225660995617604, 0.224179
93943055836, 0.2239728637583781, 0.22322364659267135, 0.22268005559529713, 0.222
26893384793267, 0.22240562017090748]

********************************************************************************
********************
The best K value is: 47

********************************************************************************
********************
```



Prediction

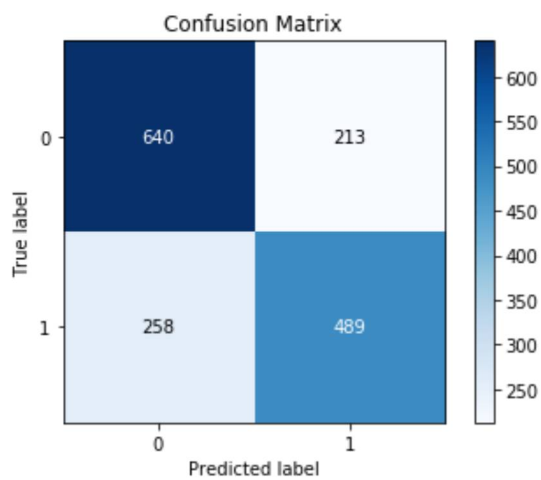
With k value 47, error is lowest.

We will use k value 47 for the prediction

In [54]:
```
knn_lib_avg = KNeighborsClassifier(n_neighbors = optimal_k_val)
knn_fit_avg = knn_lib_avg.fit(x_train_8k_avg, y_train_8k)
prediction_avg = knn_fit_avg.predict(x_test_8k_avg_t)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_8k, prediction_avg)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_8k, prediction_avg))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_8k, prediction_avg))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_8k, prediction_avg)
```

```
Accuracy score

70.5625

****************************************************************************
*******************
Classification report

             precision    recall  f1-score   support

          0       0.71      0.75      0.73       853
          1       0.70      0.65      0.67       747

   micro avg       0.71      0.71      0.71      1600
   macro avg       0.70      0.70      0.70      1600
weighted avg       0.71      0.71      0.70      1600

****************************************************************************
*******************
Confusion matrix
[[640 213]
 [258 489]]
Confusion matrix table:
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0ba4e0>



Tfidf - Word2Vec

In [55]:
```
tf_model = TfidfVectorizer()

tf_idf_matrix = tf_model.fit_transform(df_8k['Clean_Text'].values)

# we are converting a dictionary with word as a key, and the idf as a value

dicti = dict(zip(tf_model.get_feature_names(), list(tf_model.idf_)))
```

In [56]:
```
tf_sentence = []   # Empty list to store the tfidf-w2v values

for tf_sent in tqdm(w2v):
    tf_zero = np.zeros(50)     # (50,) matrix which is initial to add to the first
w2v of word in a sentence/review
    tf_count = 0                # Increases by tfidf value of previous tfidf value
for every iteration and divides the sum of tfidf-w2v of sentence/review
    for tf_word in tf_sent:
        if tf_word in w2v_word:
            tf_vec = w2v_model.wv[tf_word]  # Get tfidf_w2v for each word in a sent
ence/review
            if tf_word in dicti:
                # tf_w2v = tf_idf_matrix[row, tf_feat.index(tf_word)]
                tf_w2v = dicti[tf_word] * (tf_sent.count(tf_word))/len(tf_sent)
                tf_zero += (tf_vec * tf_w2v)    # Increase by tfidf-w2v value for e
very iteration
                tf_count += tf_w2v              # Increase by ifidf value for every
iteration
    if tf_count != 0:
        tf_zero /= tf_count                     # tfidf-w2v of sentence/review is d
ivided by total number of tfidf of words in a sentence/review (tfidf-w2v)
    tf_sentence.append(tf_zero)                # Stores all the avg w2z in an empty
list sentence
```

```
100%|████████████████████████████████████████| 8000/8000 [01:01<00:00, 129.49it/s]
```

In [57]:
```
# Normalize the data

sentence_8k_tw_norm = normalize(tf_sentence)
```

Train Test split

In [58]:
```
#Splitting into train and test data.

x_train_8k_tw, x_test_8k_tw, y_train_8k, y_test_8k = train_test_split(sentence_8k_t
w_norm, y_8k, test_size = 0.2,
                                                             random_state = 0, shuf
fle = False )
```

Time Series Split

In [59]:
```
# Import TimeSeriesSplit library
from sklearn.model_selection import TimeSeriesSplit

# Create an object
ts = TimeSeriesSplit(n_splits = 10)

for train_8k_tw, cv_8k_tw in ts.split(x_train_8k_tw):
    print(x_train_8k_tw[train_8k_tw].shape, x_train_8k_tw[cv_8k_tw].shape )
```

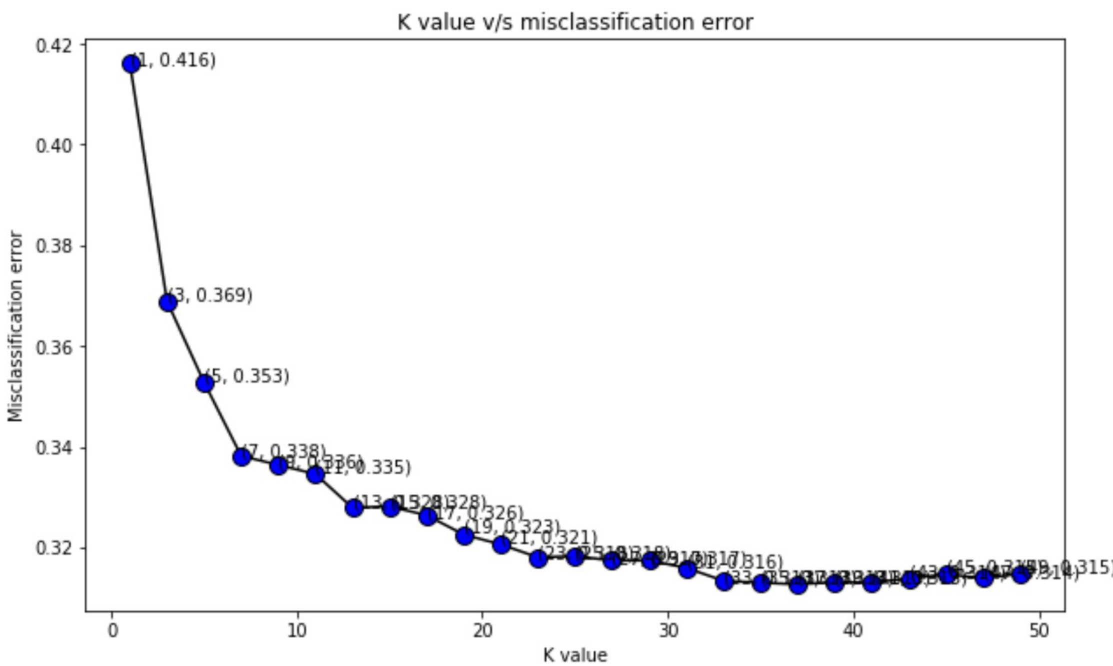
```
(590, 50) (581, 50)
(1171, 50) (581, 50)
(1752, 50) (581, 50)
(2333, 50) (581, 50)
(2914, 50) (581, 50)
(3495, 50) (581, 50)
(4076, 50) (581, 50)
(4657, 50) (581, 50)
(5238, 50) (581, 50)
(5819, 50) (581, 50)
```

Brute force with 10 fold validation

In [60]:

```
# Input the range of odd numbers

num = range(1,50,2)

cv_score = []             # It stores the cross_val_score

time = TimeSeriesSplit(n_splits=10)

for k in tqdm(num):
    knn = KNeighborsClassifier(n_neighbors = k, algorithm = 'brute', n_jobs = -1)
    score = cross_val_score(knn, x_train_8k_tw, y_train_8k, cv = time, scoring = 'r
oc_auc')
    cv_score.append(score.mean())

# Misclassification

print("Misclassification errors are:", '\n')
mse = [1-x for x in cv_score]
print(mse, '\n')
print('*'*100)

optimal_k_tw = num[mse.index(min(mse))]
print("The best K value is:" + ' ' + str(optimal_k_tw), '\n')
print('*'*100)

# To find the best K value, we can analyze from the plot (num vs misclassification
error)

plt.figure(figsize = (10,6))
plt.plot(num, mse, color = 'black', marker = 'o', markerfacecolor = 'blue', markers
ize = 10)

plt.title("K value v/s misclassification error")
plt.xlabel("K value")
plt.ylabel("Misclassification error")

# Adding annotation to get the marker position number
for xy in zip(num, np.round(mse,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)
```

```
100%|████████████████████████████████████████| 25/25 [00:51<00:00,  2.24s/it]

Misclassification errors are:

[0.4160399894244913, 0.3688653197461528, 0.352765650017943, 0.33815269396183645,
0.3363745728857547, 0.33455187783375473, 0.32786269442508, 0.32813423218847393,
0.32636813715248225, 0.32253767170442393, 0.32057558265341846, 0.318008338484716
2, 0.31827436530391995, 0.317364132806203, 0.31739297911554076, 0.31591878866275
19, 0.31336755455058785, 0.3129862862238526, 0.31264520756511316, 0.313104027807
02806, 0.31298974107606925, 0.31382787917397825, 0.3146549473006247, 0.313890083
29404655, 0.31473492853980445]

********************************************************************************
********************
The best K value is: 37

********************************************************************************
********************
```



Prediction

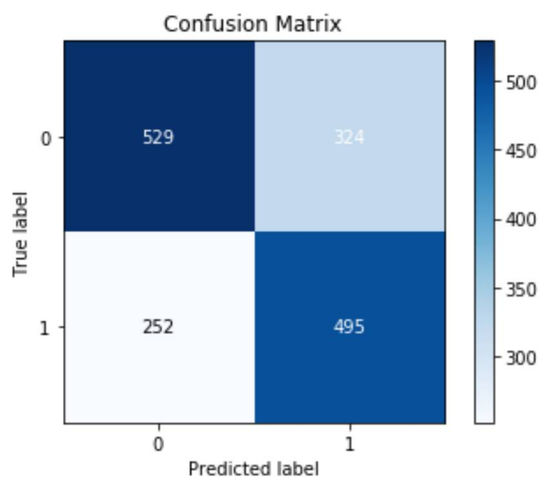
With k value 37, error is lowest

We will use k value 37 for the prediction

In [61]:
```
knn_lib_tw = KNeighborsClassifier(n_neighbors = optimal_k_tw)
knn_fit_tw = knn_lib_tw.fit(x_train_8k_tw, y_train_8k)
prediction_tw = knn_fit_tw.predict(x_test_8k_tw)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_8k, prediction_tw)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_8k, prediction_tw))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_8k, prediction_tw))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_8k, prediction_tw)
```

```
Accuracy score

64.0

****************************************************************************************************
Classification report

              precision    recall  f1-score   support

           0       0.68      0.62      0.65       853
           1       0.60      0.66      0.63       747

   micro avg       0.64      0.64      0.64      1600
   macro avg       0.64      0.64      0.64      1600
weighted avg       0.64      0.64      0.64      1600

****************************************************************************************************
Confusion matrix
[[529 324]
 [252 495]]
Confusion matrix table:
```

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x5d2327b8>



KD Tree algorithm on BOW, Tf-Idf, Avg-W2V, Tf-If-W2V

KD Tree accepts only dense matrix. For BOW and Tf-Idf vector will be in sparse matrix. So, we need convert to dense matrix.

Method-2: Simple Cross Validation

Split into train, cv and test

In [62]:
```
# Splitting into train and test data. This will be used to predict.

x_train_t, x_test_t, y_train_t, y_test_t = train_test_split(x_8k, y_8k, test_size =
0.2,
                                                            shuffle =  False, r
andom_state = 0)
```

Bag of Words

In [63]:
```
# Import CountVectorizer library
from sklearn.feature_extraction.text import CountVectorizer

# Create an instance for CountVectorizer.
# Bi-gram vector
cv = CountVectorizer(ngram_range = (1,2), min_df = 10, max_features = 500)

# Fit and transform the x_train
x_train_8k_bow_t = cv.fit_transform(x_train_t)

# Transform the x_test
x_test_8k_bow_t = cv.transform(x_test_t)

print("Type of x_train_8k_bow_t:" + ' ' + str(type(x_train_8k_bow_t)))
print("Shape of x_train_8k_bow_t:" + ' ' + str(x_train_8k_bow_t.get_shape()))
print("Number of unique words in x_train_8k_bow_t:" + ' ' + str(x_train_8k_bow_t.ge
t_shape()[1]))
```

```
Type of x_train_8k_bow_t: <class 'scipy.sparse.csr.csr_matrix'>
Shape of x_train_8k_bow_t: (6400, 500)
Number of unique words in x_train_8k_bow_t: 500
```

Normalizing and converting to dense matrix

In [64]:
```
# Normalizing the data to get everything in a single scale
# Import normalization library

from sklearn.preprocessing import normalize

x_train_8k_bow_n = normalize(x_train_8k_bow_t)
x_test_8k_bow_n = normalize(x_test_8k_bow_t)

# Converting to dense matrix
x_train_8k_bow_d = x_train_8k_bow_n.todense()
x_test_8k_bow_d = x_test_8k_bow_n.todense()

print("Type of x_train_8k_bow_d:" + ' ' +  str(type(x_train_8k_bow_d)))
print("Type of x_test_8k_bow_d:"  + ' ' + str(type(x_test_8k_bow_d)))
```

```
Type of x_train_8k_bow_d: <class 'numpy.matrixlib.defmatrix.matrix'>
Type of x_test_8k_bow_d: <class 'numpy.matrixlib.defmatrix.matrix'>
```

Split into train and cv

In [65]:
```
# Splitting train and test data into further 2 parts which we will be used to build
a model and find the best k.

x_tr_t, x_cv_t, y_tr_t, y_cv_t = train_test_split(x_train_8k_bow_d, y_train_t, test
_size = 0.2,
                                                             random_state = 0, shuffl
e = False)
```

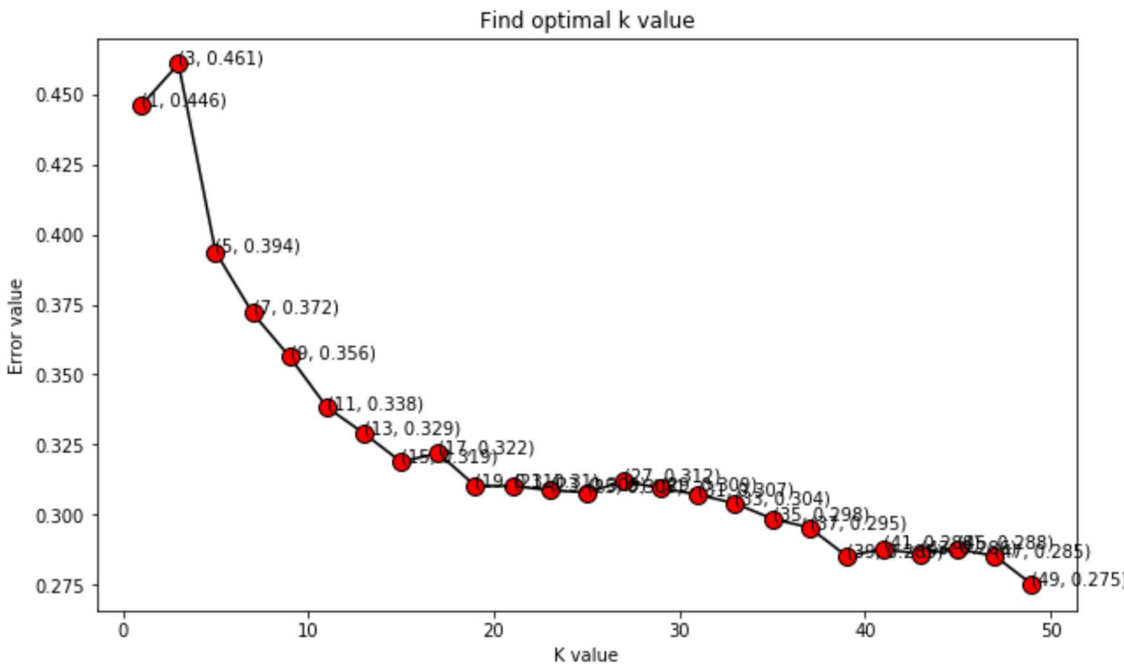
Find the best K value

In [66]:
```
error_bow = []                                                          # error value is
stored in this list

# Passing only odd k value

k = range(1,50,2)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i, algorithm = 'kd_tree', n_jobs = -1)
    knn = knn.fit(x_tr_t, y_tr_t)
    pred = knn.predict(x_cv_t)
    error_bow.append(np.mean(pred != y_cv_t))

    accuracy = accuracy_score(y_cv_t, pred, normalize = True)*100
    print("CV Accuracy for k value %d is %d%%:" % (i, accuracy) )
```

```
CV Accuracy for k value 1 is 55%:
CV Accuracy for k value 3 is 53%:
CV Accuracy for k value 5 is 60%:
CV Accuracy for k value 7 is 62%:
CV Accuracy for k value 9 is 64%:
CV Accuracy for k value 11 is 66%:
CV Accuracy for k value 13 is 67%:
CV Accuracy for k value 15 is 68%:
CV Accuracy for k value 17 is 67%:
CV Accuracy for k value 19 is 68%:
CV Accuracy for k value 21 is 68%:
CV Accuracy for k value 23 is 69%:
CV Accuracy for k value 25 is 69%:
CV Accuracy for k value 27 is 68%:
CV Accuracy for k value 29 is 69%:
CV Accuracy for k value 31 is 69%:
CV Accuracy for k value 33 is 69%:
CV Accuracy for k value 35 is 70%:
CV Accuracy for k value 37 is 70%:
CV Accuracy for k value 39 is 71%:
CV Accuracy for k value 41 is 71%:
CV Accuracy for k value 43 is 71%:
CV Accuracy for k value 45 is 71%:
CV Accuracy for k value 47 is 71%:
CV Accuracy for k value 49 is 72%:
```

In [67]:
```
#Let's verify using plot

plt.figure(figsize = (10,6))
plt.plot(range(1,50,2), error_bow, marker = 'o', color = 'black', markerfacecolor =
'r', markersize = '10')

# Annotation:
for xy in zip(k, np.round(error_bow,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)

plt.title("Find optimal k value")
plt.xlabel("K value")
plt.ylabel("Error value")
```

Out[67]: Text(0, 0.5, 'Error value')



Observation:

From the prediction based on train and CV split, we got 49 as best k value with an accuracy of 72%

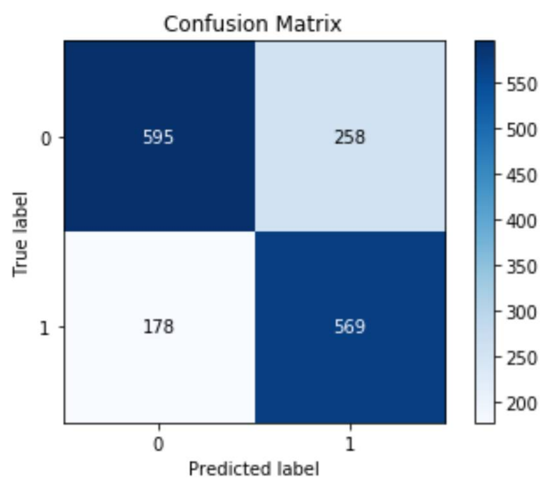
Graphically also, we got same

In [86]:
```
knn = KNeighborsClassifier(n_neighbors = 49)
knn_fit = knn.fit(x_train_8k_bow_d, y_train_t)
prediction = knn_fit.predict(x_test_8k_bow_d)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_t, prediction)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_t, prediction))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_t, prediction))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_t, prediction)
```

```
Accuracy score

72.75

****************************************************************************************************
Classification report

              precision    recall  f1-score   support

           0       0.77      0.70      0.73       853
           1       0.69      0.76      0.72       747

   micro avg       0.73      0.73      0.73      1600
   macro avg       0.73      0.73      0.73      1600
weighted avg       0.73      0.73      0.73      1600

****************************************************************************************************
Confusion matrix
[[595 258]
 [178 569]]
Confusion matrix table:
```

Out[86]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x43eba7f0>
```



Observation:

While building a model we got an accuracy of 72%

When we tested on the test set, we got 72.75% which is very close to 7%

Tf-Idf

In [69]:
```
# Import Tf-Idf
from sklearn.feature_extraction.text import TfidfVectorizer

# Create an instance
tf = TfidfVectorizer(ngram_range = (1,2), min_df = 10, max_features = 500)

# Fit and transform the x_train
x_train_tf_t = tf.fit_transform(x_train_t)

# Transform the x_test
x_test_tf_t = tf.transform(x_test_t)

print("Type of x_train_tf_t:" + ' ' + str(type(x_train_tf_t)))
print("Shape of x_train_tf_t:" + ' ' + str(x_train_tf_t.get_shape()))
print("Number of unique words in x_train_8k_tf:" + ' ' + str(x_train_tf_t.get_shape
()[1]))
```

```
Type of x_train_tf_t: <class 'scipy.sparse.csr.csr_matrix'>
Shape of x_train_tf_t: (6400, 500)
Number of unique words in x_train_8k_tf: 500
```

Normalizing and converting to dense matrix

In [70]:
```
# Normalizing the data to get everything in a single scale
# Import normalization library

from sklearn.preprocessing import normalize

x_train_tf_n = normalize(x_train_tf_t)
x_test_tf_n = normalize(x_test_tf_t)

# Converting to dense matrix
x_train_tf_d = x_train_tf_n.todense()
x_test_tf_d = x_test_tf_n.todense()

print("Type of x_train_tf_d:" + ' ' +  str(type(x_train_tf_d)))
print("Type of x_test_tf_d:"  + ' ' + str(type(x_test_tf_d)))
```

```
Type of x_train_tf_d: <class 'numpy.matrixlib.defmatrix.matrix'>
Type of x_test_tf_d: <class 'numpy.matrixlib.defmatrix.matrix'>
```

split into train and cv

In [71]:
```
# Splitting train and test data into further 2 parts which we will be used to build
a model and find the best k.

x_tr_tf_t, x_cv_tf_t, y_tr_tf_t, y_cv_tf_t = train_test_split(x_train_tf_d, y_train
_t, test_size = 0.2,
                                                            random_state = 0, shuffl
e = False)
```

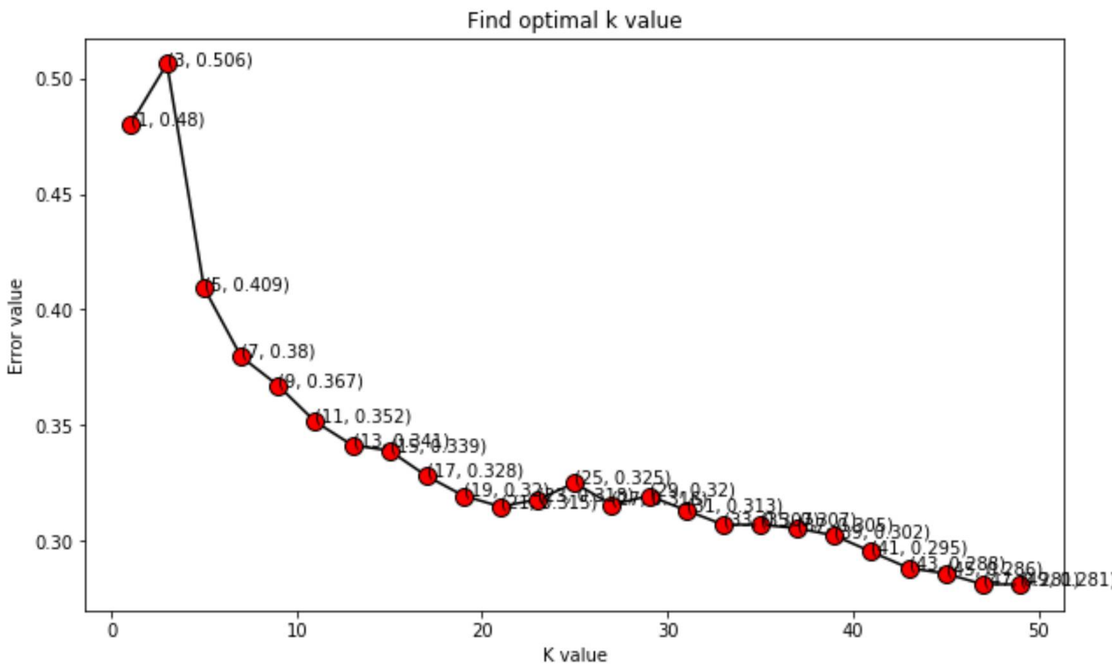
Find the best K value

In [72]:
```
error_tf = []                                                    # error value is s
tored in this list

# Passing only odd k value

k = range(1,50,2)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i, algorithm = 'kd_tree', n_jobs = -1)
    knn = knn.fit(x_tr_tf_t, y_tr_tf_t)
    pred = knn.predict(x_cv_tf_t)
    error_tf.append(np.mean(pred != y_cv_tf_t))

    accuracy = accuracy_score(y_cv_tf_t, pred, normalize = True)*100
    print("CV Accuracy for k value %d is %d%%:" % (i, accuracy) )
```

```
CV Accuracy for k value 1 is 52%:
CV Accuracy for k value 3 is 49%:
CV Accuracy for k value 5 is 59%:
CV Accuracy for k value 7 is 62%:
CV Accuracy for k value 9 is 63%:
CV Accuracy for k value 11 is 64%:
CV Accuracy for k value 13 is 65%:
CV Accuracy for k value 15 is 66%:
CV Accuracy for k value 17 is 67%:
CV Accuracy for k value 19 is 68%:
CV Accuracy for k value 21 is 68%:
CV Accuracy for k value 23 is 68%:
CV Accuracy for k value 25 is 67%:
CV Accuracy for k value 27 is 68%:
CV Accuracy for k value 29 is 68%:
CV Accuracy for k value 31 is 68%:
CV Accuracy for k value 33 is 69%:
CV Accuracy for k value 35 is 69%:
CV Accuracy for k value 37 is 69%:
CV Accuracy for k value 39 is 69%:
CV Accuracy for k value 41 is 70%:
CV Accuracy for k value 43 is 71%:
CV Accuracy for k value 45 is 71%:
CV Accuracy for k value 47 is 71%:
CV Accuracy for k value 49 is 71%:
```

In [73]:
```python
#Let's verify using plot

plt.figure(figsize = (10,6))
plt.plot(range(1,50,2), error_tf, marker = 'o', color = 'black', markerfacecolor = 
'r', markersize = '10')

# Annotation:
for xy in zip(k, np.round(error_tf,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)

plt.title("Find optimal k value")
plt.xlabel("K value")
plt.ylabel("Error value")
```

Out[73]:
```
Text(0, 0.5, 'Error value')
```



Observation:

From the prediction based on train and CV split, we got 43, 45, 47 and 49 as best k value with an accuracy of 71%

Graphically also, we got same

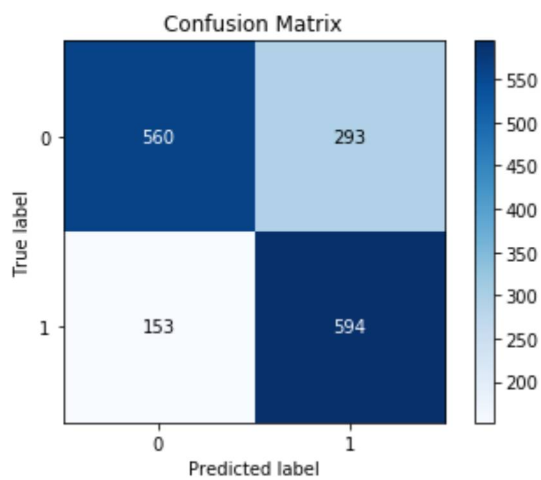
Prediction

With k value either 43, 45, 47 or 49, we will predict the test data

In [74]:
```
knn = KNeighborsClassifier(n_neighbors = 45)
knn_fit = knn.fit(x_train_tf_d, y_train_t)
prediction = knn_fit.predict(x_test_tf_d)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_t, prediction)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_t, prediction))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_t, prediction))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_t, prediction)
```

```
Accuracy score

72.125

****************************************************************************************
*******************
Classification report

              precision    recall  f1-score   support

           0       0.79      0.66      0.72       853
           1       0.67      0.80      0.73       747

   micro avg       0.72      0.72      0.72      1600
   macro avg       0.73      0.73      0.72      1600
weighted avg       0.73      0.72      0.72      1600

****************************************************************************************
*******************
Confusion matrix
[[560 293]
 [153 594]]
Confusion matrix table:
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x558dd550>



Observation:

While building a model we got an accuracy of 71%

When we tested on the test set, we got 72.12% which is very close to 71%

Avg Word2Vec

In [75]:
```
# We have already obtained avg Word2Vec and assigned to sentence_8k_tw_norm
# Let's check the type of it.

type(sentence_8k_avg_norm)
```

Out[75]: numpy.ndarray

Since it is already in array, we don't need to convert it to dense matrix.

Split into train, test and cv

In [76]:
```
#Splitting sentence_8k_avg_norm into train and test data.

x_train_avg_t, x_test_avg_t, y_train_t, y_test_t = train_test_split(sentence_8k_avg
_norm, y_8k, test_size = 0.2,
                                                             random_state = 0, shuf
fle = False )


# Splitting into further 2 parts train and cv

x_tr_avg, x_cv_avg, y_tr_avg, y_cv_avg = train_test_split(x_train_avg_t, y_train_t,
test_size = 0.2,
                                                             random_state = 0, shuf
fle = False )
```

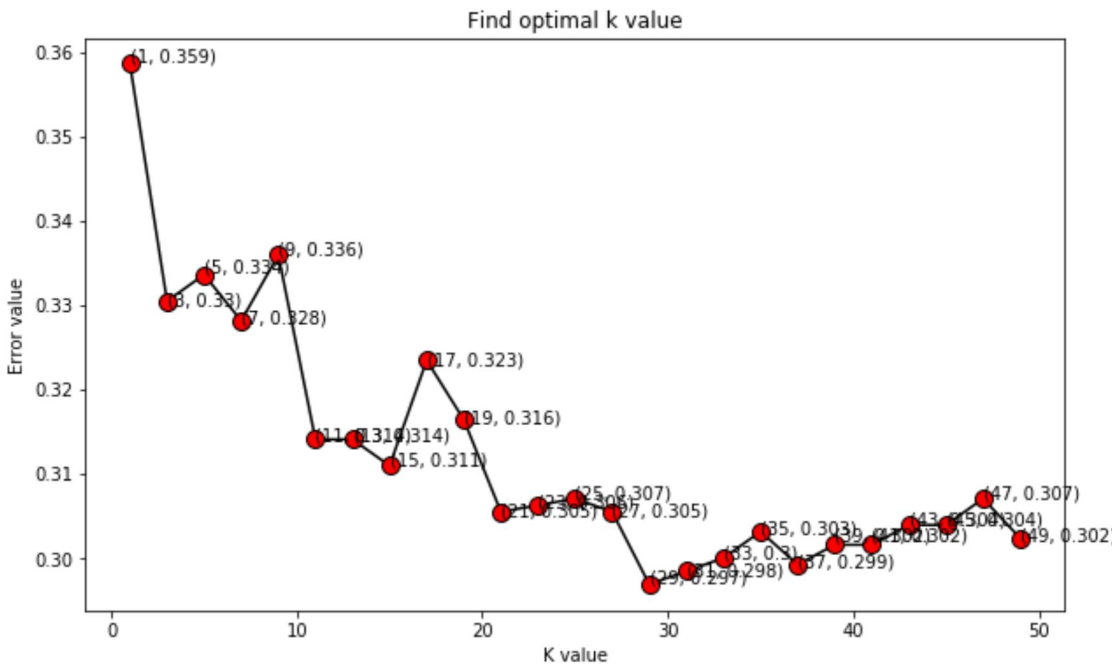
Find the best K value

In [77]:
```
error_avg = []                                                     # error value is
stored in this list

# Passing only odd k value

k = range(1,50,2)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i, algorithm = 'kd_tree', n_jobs = -1)
    knn = knn.fit(x_tr_avg, y_tr_avg)
    pred = knn.predict(x_cv_avg)
    error_avg.append(np.mean(pred != y_cv_avg))

    accuracy = accuracy_score(y_cv_avg, pred, normalize = True)*100
    print("CV Accuracy for k value %d is %d%%:" % (i, accuracy) )
```

```
CV Accuracy for k value 1 is 64%:
CV Accuracy for k value 3 is 66%:
CV Accuracy for k value 5 is 66%:
CV Accuracy for k value 7 is 67%:
CV Accuracy for k value 9 is 66%:
CV Accuracy for k value 11 is 68%:
CV Accuracy for k value 13 is 68%:
CV Accuracy for k value 15 is 68%:
CV Accuracy for k value 17 is 67%:
CV Accuracy for k value 19 is 68%:
CV Accuracy for k value 21 is 69%:
CV Accuracy for k value 23 is 69%:
CV Accuracy for k value 25 is 69%:
CV Accuracy for k value 27 is 69%:
CV Accuracy for k value 29 is 70%:
CV Accuracy for k value 31 is 70%:
CV Accuracy for k value 33 is 70%:
CV Accuracy for k value 35 is 69%:
CV Accuracy for k value 37 is 70%:
CV Accuracy for k value 39 is 69%:
CV Accuracy for k value 41 is 69%:
CV Accuracy for k value 43 is 69%:
CV Accuracy for k value 45 is 69%:
CV Accuracy for k value 47 is 69%:
CV Accuracy for k value 49 is 69%:
```

In [78]:
```
#Let's verify using plot

plt.figure(figsize = (10,6))
plt.plot(range(1,50,2), error_avg, marker = 'o', color = 'black', markerfacecolor =
'r', markersize = '10')

# Annotation:
for xy in zip(k, np.round(error_avg,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)

plt.title("Find optimal k value")
plt.xlabel("K value")
plt.ylabel("Error value")
```

Out[78]: Text(0, 0.5, 'Error value')



Observation:

From the prediction based on train and CV split, we got 29, 31 and 33 as best k value with an accuracy of 70%

Graphically also, we got same

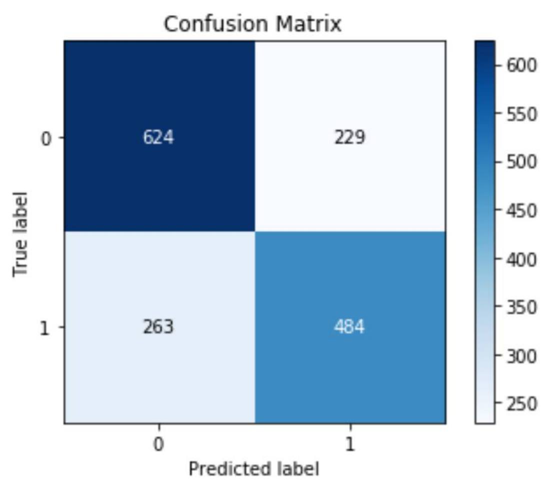
Prediction

With k value as 31, we will predict the test set.

In [79]:
```
knn = KNeighborsClassifier(n_neighbors = 31)
knn_fit = knn.fit(x_train_avg_t, y_train_t)
prediction = knn_fit.predict(x_test_avg_t)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_t, prediction)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_t, prediction))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_t, prediction))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_t, prediction)
```

```
Accuracy score

69.25

*******************************************************************************
*******************
Classification report

              precision    recall  f1-score   support

           0       0.70      0.73      0.72       853
           1       0.68      0.65      0.66       747

   micro avg       0.69      0.69      0.69      1600
   macro avg       0.69      0.69      0.69      1600
weighted avg       0.69      0.69      0.69      1600

*******************************************************************************
*******************
Confusion matrix
[[624 229]
 [263 484]]
Confusion matrix table:
```

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x5d7e94e0>



Observation:

While building a model we got an accuracy of 70%

When we tested on the test set, we got 69.25% which is very close to 70%

Tf-Idf Word2Vec

In [80]:
```
# We have already obtained Tf-Idf Word2Vec and assigned to sentence_8k_tw_norm
# Let's check the type of it.

type(sentence_8k_tw_norm)
```

Out[80]: numpy.ndarray

Split into train, test and cv

In [81]:
```
#Splitting sentence_8k_avg_norm into train and test data.

x_train_tw_t, x_test_tw_t, y_train_t, y_test_t = train_test_split(sentence_8k_tw_no
rm, y_8k, test_size = 0.2,
                                                              random_state = 0, shuf
fle = False )


# Splitting into further 2 parts train and cv

x_tr_tw, x_cv_tw, y_tr_tw, y_cv_tw = train_test_split(x_train_tw_t, y_train_t, test
_size = 0.2,
                                                              random_state = 0, shuf
fle = False )
```

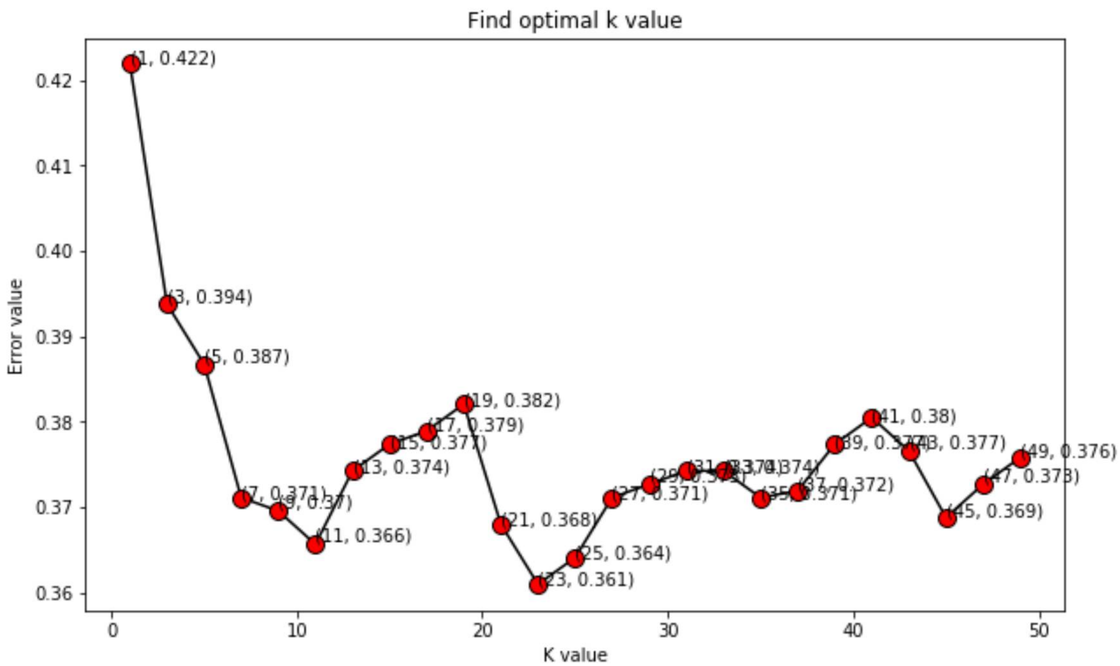
Find the best K value

In [82]:
```
error_tw = []                                                    # error value is s
tored in this list

# Passing only odd k value

k = range(1,50,2)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i, algorithm = 'kd_tree', n_jobs = -1)
    knn = knn.fit(x_tr_tw, y_tr_tw)
    pred = knn.predict(x_cv_tw)
    error_tw.append(np.mean(pred != y_cv_tw))

    accuracy = accuracy_score(y_cv_tw, pred, normalize = True)*100
    print("CV Accuracy for k value %d is %d%%:" % (i, accuracy) )
```

```
CV Accuracy for k value 1 is 57%:
CV Accuracy for k value 3 is 60%:
CV Accuracy for k value 5 is 61%:
CV Accuracy for k value 7 is 62%:
CV Accuracy for k value 9 is 63%:
CV Accuracy for k value 11 is 63%:
CV Accuracy for k value 13 is 62%:
CV Accuracy for k value 15 is 62%:
CV Accuracy for k value 17 is 62%:
CV Accuracy for k value 19 is 61%:
CV Accuracy for k value 21 is 63%:
CV Accuracy for k value 23 is 63%:
CV Accuracy for k value 25 is 63%:
CV Accuracy for k value 27 is 62%:
CV Accuracy for k value 29 is 62%:
CV Accuracy for k value 31 is 62%:
CV Accuracy for k value 33 is 62%:
CV Accuracy for k value 35 is 62%:
CV Accuracy for k value 37 is 62%:
CV Accuracy for k value 39 is 62%:
CV Accuracy for k value 41 is 61%:
CV Accuracy for k value 43 is 62%:
CV Accuracy for k value 45 is 63%:
CV Accuracy for k value 47 is 62%:
CV Accuracy for k value 49 is 62%:
```

In [83]:
```
#Let's verify using plot

plt.figure(figsize = (10,6))
plt.plot(range(1,50,2), error_tw, marker = 'o', color = 'black', markerfacecolor =
'r', markersize = '10')

# Annotation:
for xy in zip(k, np.round(error_tw,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy)

plt.title("Find optimal k value")
plt.xlabel("K value")
plt.ylabel("Error value")
```

Out[83]:
```
Text(0, 0.5, 'Error value')
```



Observation:

From the prediction based on train and CV split, we got 23 as best k value with an accuracy of 63%

Graphically also, we got same

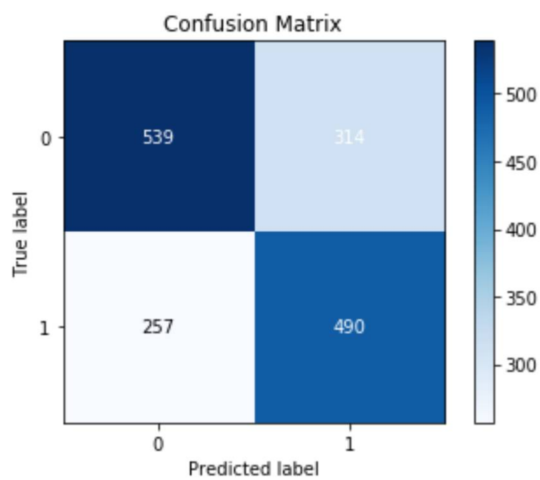
Prediction

With k value as 23, we will predict the test set.

In [87]:
```
knn = KNeighborsClassifier(n_neighbors = 23)
knn_fit = knn.fit(x_train_tw_t, y_train_t)
prediction = knn_fit.predict(x_test_tw_t)

# Get the metrics

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy score", '\n')
print(accuracy_score(y_test_t, prediction)*100, '\n')
print('*'*100)

print("Classification report", '\n')
print(classification_report(y_test_t, prediction))
print('*'*100)

print("Confusion matrix")
print(confusion_matrix(y_test_t, prediction))

# Referred from: https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
print("Confusion matrix table:", '\n')
skplt.metrics.plot_confusion_matrix(y_test_t, prediction)
```

```
Accuracy score

64.3125

****************************************************************************
*******************
Classification report

              precision    recall  f1-score   support

           0       0.68      0.63      0.65       853
           1       0.61      0.66      0.63       747

   micro avg       0.64      0.64      0.64      1600
   macro avg       0.64      0.64      0.64      1600
weighted avg       0.65      0.64      0.64      1600

****************************************************************************
*******************
Confusion matrix
[[539 314]
 [257 490]]
Confusion matrix table:
```

Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x563054e0>



Observation:

While building a model we got an accuracy of 63%

When we tested on the test set, we got 64% which is very close to 63%

Summary on pretty table

In [88]:
```
from prettytable import PrettyTable

a = PrettyTable()


a.field_names = ["S.No", "Featurization", "K", "Accuracy", "Precision", "Recall", "
f1-score"]

a.add_row([ (1), "Bag of Words", 49, 74.5, 0.74, 0.74, 0.74 ])
a.add_row([ (2), "Tf-Idf", 49, 76.56, 0.77, 0.77, 0.75])
a.add_row([ (3), "Avg Word2Vec", 37, 70.56, 0.71, 0.68, 0.70 ])
a.add_row([ (4), "Tf-Idf Word2Vec", 47, 64, 0.64, 0.64, 0.64 ])

print(a.get_string(title = "Summary Table for Brute Force algorithm"))

from prettytable import PrettyTable

b = PrettyTable()

b.field_names = ["S.No", "Featurization", "K", "Accuracy", "Precision", "Recall", "
f1-score"]

b.add_row([ 1, "Bag of Words", 49, 72.72, 0.74, 0.73, 0.73])
b.add_row([ 2, "Tf-Idf", 45, 72.12, 0.73, 0.73, 0.73])
b.add_row([ 3, "Avg Word2Vec",  31, 69.25, 0.69, 0.69, 0.69 ])
b.add_row([ 4, "Tf-Idf Word2Vec", 23, 64.31, 0.64, 0.64, 0.64 ])

print(b.get_string(title = "Summary Table for KD Tree algorithm"))
```

```
+-------------------------------------------------------------------------+
|                 Summary Table for Brute Force algorithm                 |
+------+-----------------+----+----------+-----------+--------+----------+
| S.No |  Featurization  | K  | Accuracy | Precision | Recall | f1-score |
+------+-----------------+----+----------+-----------+--------+----------+
|  1   |   Bag of Words  | 49 |   74.5   |    0.74   |  0.74  |   0.74   |
|  2   |      Tf-Idf     | 49 |  76.56   |    0.77   |  0.77  |   0.75   |
|  3   |   Avg Word2Vec  | 37 |  70.56   |    0.71   |  0.68  |   0.7    |
|  4   | Tf-Idf Word2Vec | 47 |    64    |    0.64   |  0.64  |   0.64   |
+------+-----------------+----+----------+-----------+--------+----------+
+-------------------------------------------------------------------------+
|                   Summary Table for KD Tree algorithm                   |
+------+-----------------+----+----------+-----------+--------+----------+
| S.No |  Featurization  | K  | Accuracy | Precision | Recall | f1-score |
+------+-----------------+----+----------+-----------+--------+----------+
|  1   |   Bag of Words  | 49 |  72.72   |    0.74   |  0.73  |   0.73   |
|  2   |      Tf-Idf     | 45 |  72.12   |    0.73   |  0.73  |   0.73   |
|  3   |   Avg Word2Vec  | 31 |  69.25   |    0.69   |  0.69  |   0.69   |
|  4   | Tf-Idf Word2Vec | 23 |  64.31   |    0.64   |  0.64  |   0.64   |
+------+-----------------+----+----------+-----------+--------+----------+
```

Conclusion:

1) Accuracy for both brute force algorithm and kd tree algorithm are almost similar. Both gave almost same result.

2) There could be 2 reasons for the accuracy being very low.

(a) Datapoints low: As we picked low datapoints, it may be that, model couldn't get enough information to read and predict.

(b) KNN method: It may be that KNN is not suitable for this dataset to predict with better accuracy.

In [ ]: