Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [121]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

```python
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:

```python
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 da
ta points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3
LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 L
IMIT 5000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a
negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (5000, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [3]:

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:

```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUN
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when	2

						considering the price...	
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...

In [6]:
```
display['COUNT(*)'].sum()
```

Out[6]:
```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2

4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inp
lace=False, kind='quicksort', na_position='last')
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"
}, keep='first', inplace=False)
final.shape
```

Out[9]:
```
(4986, 10)
```

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 99.72

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulne
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entrie
s left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(4986, 10)
```

Out[13]:
```
1    4178
0     808
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [14]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
```

```python
print(sent_4900)
print("="*50)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />
http://www.amazon.com/VICTOR-FLY-MAGNET-BAIT-REFILL/dp/B00004RBDY<br /><br /
>The Victor M380 and M502 traps are unreal, of course -- total fly genocide.
Pretty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these
chips are.  The best thing was that there were a lot of "brown" chips in the
bsg (my favorite), so I bought some more through amazon and shared with fami
ly and friends.  I am a little disappointed that there are not, so far, very
many brown chips in these bags, but the flavor is still very good.  I like t
hem better than the yogurt and green onion flavor because they do not seem t
o be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are t
hicker and crunchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they wer
e ordering; the other wants crispy cookies.  Hey, I'm sorry; but these revie
ws do nobody any good beyond reminding us to look  before ordering.<br /><br
/>These are chocolate-oatmeal cookies.  If you don't like that combination,
don't order this type of cookie.  I find the combo quite nice, really.  The
oatmeal sort of "calms" the rich chocolate flavor and gives the cookie sort
of a coconut-type consistency.  Now let's also remember that tastes differ;
so, I've given my opinion.<br /><br />Then, these are soft, chewy cookies --
as advertised.  They are not "crispy" cookies, or the blurb would say "crisp
y," rather than "chewy."  I happen to like raw cookie dough; however, I do
n't see where these taste like raw cookie dough.  Both are soft, however, so
is this the confusion?  And, yes, they stick together.  Soft cookies tend to
do that.  They aren't individually wrapped, which would add to the cost.  Oh
yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br />So, if yo
u want something hard and crisp, I suggest Nabiso's Ginger Snaps.  If you wa
nt a cookie that's soft, chewy and tastes like a combination of chocolate an
d oatmeal, give these a try.  I'm here to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k c
up is great coffee.  dcaf is very good as well
==================================================
```

In [15]:

```python
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)
```

```
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />
/><br />The Victor M380 and M502 traps are unreal, of course -- total fly ge
nocide. Pretty stinky, but only right nearby.
```

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-rem
ove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
Why is this $[...] when the same product is available for $[...] here? />The
Victor M380 and M502 traps are unreal, of course -- total fly genocide. Pret
ty stinky, but only right nearby.
==================================================
I recently tried this flavor/brand and was surprised at how delicious these
chips are.  The best thing was that there were a lot of "brown" chips in the
bsg (my favorite), so I bought some more through amazon and shared with fami
ly and friends.  I am a little disappointed that there are not, so far, very
many brown chips in these bags, but the flavor is still very good.  I like t
hem better than the yogurt and green onion flavor because they do not seem t
o be as salty, and the onion flavor is better.  If you haven't eaten Kettle
chips before, I recommend that you try a bag before buying bulk.  They are t
hicker and crunchier than Lays but just as fresh out of the bag.
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they wer
e ordering; the other wants crispy cookies.  Hey, I'm sorry; but these revie
```

```
ws do nobody any good beyond reminding us to look  before ordering.These are
chocolate-oatmeal cookies.  If you don't like that combination, don't order
this type of cookie.  I find the combo quite nice, really.  The oatmeal sort
of "calms" the rich chocolate flavor and gives the cookie sort of a coconut-
type consistency.  Now let's also remember that tastes differ; so, I've give
n my opinion.Then, these are soft, chewy cookies -- as advertised.  They are
not "crispy" cookies, or the blurb would say "crispy," rather than "chewy."
I happen to like raw cookie dough; however, I don't see where these taste li
ke raw cookie dough.  Both are soft, however, so is this the confusion?  An
d, yes, they stick together.  Soft cookies tend to do that.  They aren't ind
ividually wrapped, which would add to the cost.  Oh yeah, chocolate chip coo
kies tend to be somewhat sweet.So, if you want something hard and crisp, I s
uggest Nabiso's Ginger Snaps.  If you want a cookie that's soft, chewy and t
astes like a combination of chocolate and oatmeal, give these a try.  I'm he
re to place my second order.
==================================================
love to order my coffee on amazon.  easy and shows up quickly.This k cup is
great coffee.  dcaf is very good as well
```

In [17]:

```python
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [18]:

```python
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they wer
e ordering; the other wants crispy cookies.  Hey, I am sorry; but these revi
ews do nobody any good beyond reminding us to look  before ordering.<br /><b
```

```
r />These are chocolate-oatmeal cookies.  If you do not like that combinatio
n, do not order this type of cookie.  I find the combo quite nice, really.
The oatmeal sort of "calms" the rich chocolate flavor and gives the cookie s
ort of a coconut-type consistency.  Now let is also remember that tastes dif
fer; so, I have given my opinion.<br /><br />Then, these are soft, chewy coo
kies -- as advertised.  They are not "crispy" cookies, or the blurb would sa
y "crispy," rather than "chewy."  I happen to like raw cookie dough; howeve
r, I do not see where these taste like raw cookie dough.  Both are soft, how
ever, so is this the confusion?  And, yes, they stick together.  Soft cookie
s tend to do that.  They are not individually wrapped, which would add to th
e cost.  Oh yeah, chocolate chip cookies tend to be somewhat sweet.<br /><br
/>So, if you want something hard and crisp, I suggest Nabiso is Ginger Snap
s.  If you want a cookie that is soft, chewy and tastes like a combination o
f chocolate and oatmeal, give these a try.  I am here to place my second ord
er.
==================================================
```

In [19]:

```python
#remove words with numbers python: https://stackoverflow.com/a/18082370/408403
9
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />
/><br />The Victor  and  traps are unreal, of course -- total fly genocide.
Pretty stinky, but only right nearby.
```

In [20]:

```python
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were ord
ering the other wants crispy cookies Hey I am sorry but these reviews do nob
ody any good beyond reminding us to look before ordering br br These are cho
colate oatmeal cookies If you do not like that combination do not order this
type of cookie I find the combo quite nice really The oatmeal sort of calms
the rich chocolate flavor and gives the cookie sort of a coconut type consis
tency Now let is also remember that tastes differ so I have given my opinion
br br Then these are soft chewy cookies as advertised They are not crispy co
okies or the blurb would say crispy rather than chewy I happen to like raw c
ookie dough however I do not see where these taste like raw cookie dough Bot
h are soft however so is this the confusion And yes they stick together Soft
cookies tend to do that They are not individually wrapped which would add to
the cost Oh yeah chocolate chip cookies tend to be somewhat sweet br br So i
f you want something hard and crisp I suggest Nabiso is Ginger Snaps If you
want a cookie that is soft chewy and tastes like a combination of chocolate
and oatmeal give these a try I am here to place my second order
```

In [21]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
```

```
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████| 4986/4986 [00:09<00:00, 512.71it/s]
```

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry reviews nobody good beyond reminding us look ordering chocolate oatmeal cookies not like combination not order type cookie find combo quite nice really oatmeal sort calms rich chocolate flavor gives cookie sort coconut type consistency let also remember tastes differ given opinion soft chewy cookies advertised not crispy cookies blurb would say crispy rather chewy happen like raw cookie dough however not see taste like raw cookie dough soft however confusion yes stick together soft cookies tend not individually wrapped would add cost oh yeah chocolate chip cookies tend somewhat sweet want something hard crisp suggest nabiso ginger snaps want cookie soft chewy tastes like combination chocolate oatmeal give try place second order'
```

[3.2] Preprocessing Review Summary

In [24]:

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

In [25]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
```

```
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott',
'abby', 'abdominal', 'abiding', 'ability']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

[4.2] Bi-Grams and n-Grams.

In [26]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/sta
ble/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final
_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.3] TF-IDF

In [27]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_featu
re_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
```

```
print("the number of unique words including both unigrams and bigrams ", final
_tf_idf.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able f
ind', 'able get', 'absolute', 'absolutely', 'absolutely delicious', 'absolut
ely love', 'absolutely no', 'according']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

[4.4] Word2Vec

In [28]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [29]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZP
Y
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
```

```
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negati
ve300.bin', binary=True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v =
True, to train your own w2v ")
```

```
[('alternative', 0.994074285030365), ('snack', 0.9932457208633423), ('health
y', 0.9926791191101074), ('want', 0.992630660533905), ('brownies', 0.9923517
70401001), ('satisfying', 0.992313802242279), ('tasty', 0.9920635223388672),
('gatorade', 0.9919566512107849), ('licorice', 0.9919050931930542), ('cris
p', 0.991763710975647)]
==================================================
[('popcorn', 0.9995943307876587), ('varieties', 0.9994621872901917), ('lay
s', 0.9994592666625977), ('wow', 0.9993892312049866), ('individual', 0.99937
99924850464), ('awful', 0.9993244409561157), ('world', 0.99930739402771),
('become', 0.9992760419845581), ('de', 0.9992551803588867), ('clear', 0.9992
419481277466)]
```

In [30]:

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stink
y', 'right', 'nearby', 'used', 'ca', 'not', 'beat', 'great', 'received', 'sh
ipment', 'could', 'hardly', 'wait', 'try', 'love', 'call', 'instead', 'remov
ed', 'easily', 'daughter', 'designed', 'printed', 'use', 'car', 'windows',
'beautifully', 'shop', 'program', 'going', 'lot', 'fun', 'everywhere', 'lik
e', 'tv', 'computer', 'really', 'good', 'idea', 'final', 'outstanding', 'win
dow', 'everybody', 'asks', 'bought', 'made']
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

```
In [31]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
         st
         for sent in tqdm(list_of_sentance): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might
         need to change this to 300 if you use google's w2v
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words:
                     vec = w2v_model.wv[word]
                     sent_vec += vec
                     cnt_words += 1
             if cnt_words != 0:
                 sent_vec /= cnt_words
             sent_vectors.append(sent_vec)
         print(len(sent_vectors))
         print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████| 4986/4986 [00:29<00:00, 166.43i
t/s]
```

```
4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [32]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
         model = TfidfVectorizer()
         tf_idf_matrix = model.fit_transform(preprocessed_reviews)
         # we are converting a dictionary with word as a key, and the idf as a value
         dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [33]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
          = tfidf

         tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
         this list
         row=0;
         for sent in tqdm(list_of_sentance): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
```

```
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████| 4986/4986 [02:32<00:00, 53.58it/s]
```

[5] Assignment 3: KNN

1. **Apply Knn(brute force version) on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Apply Knn(kd tree version) on these feature sets**
   NOTE: sklearn implementation of kd-tree accepts only dense matrices, you need to convert the sparse matrices of CountVectorizer/TfidfVectorizer into dense matices. You can convert sparse matrices to dense using .toarray() attribute. For more information please visit this [link]

    - SET 5:Review text, preprocessed one converted into vectors using (BOW) but with restriction on maximum features generated.

      ```
      count_vect = CountVectorizer(min_df=10, max_features=500)
      count_vect.fit(preprocessed_reviews)
      ```

- SET 6:Review text, preprocessed one converted into vectors using (TFIDF) but with restriction on maximum features generated.

```
tf_idf_vect = TfidfVectorizer(min_df=10, max_features=500)
tf_idf_vect.fit(preprocessed_reviews)
```

- SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
- SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

3. **The hyper paramter tuning(find best K)**

   - Find the best hyper parameter which will give the maximum [AUC] value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

4. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

5. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.

2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

[5.1] Applying KNN brute force

In [219]:
```
labels=final['Score'].iloc[0:4986]
labels.value_counts()
new_reviews=pd.Series(preprocessed_reviews)
type(new_reviews)
```

Out[219]:
```
pandas.core.series.Series
```

In [222]:
```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, test_size=0.3, random_state=0)
print(len(x_train), len(x_test), len(y_train), len(y_test))
```

```
3490 1496 3490 1496
```

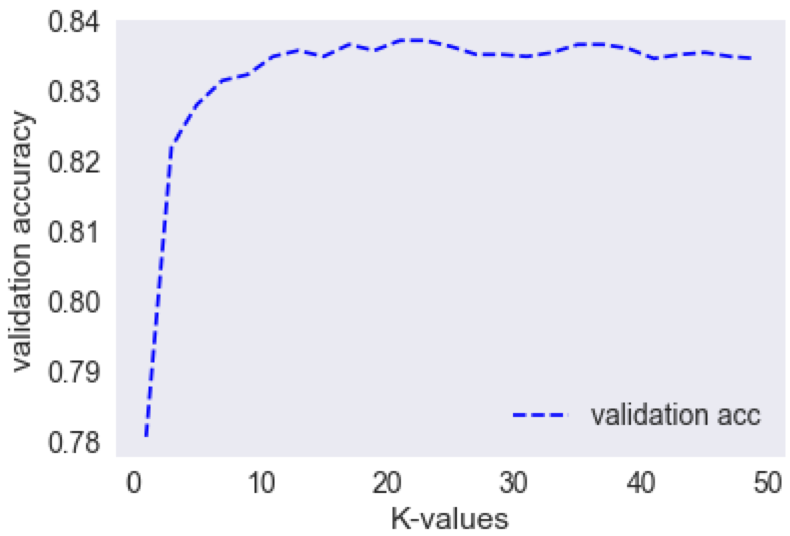
[5.1.1] Applying KNN brute force on BOW, SET 1

In [227]:
```
from sklearn.neighbors import KNeighborsClassifier
count_vect = CountVectorizer(ngram_range=(1,1))
train_reviews=count_vect.fit_transform(x_train)
test_reviews=count_vect.transform(x_test)
```

BOW using K-fold cross validation

In [229]:
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
```

```
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 21
```

In [230]:

```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn_model.fit(train_reviews, y_train)
predict=knn_model.predict(test_reviews)
acc=accuracy_score(y_test, predict)*100
```

```
test accuracy=85.36096256684492
```

Finding Hyper parameter using area under roc curve

In [231]:

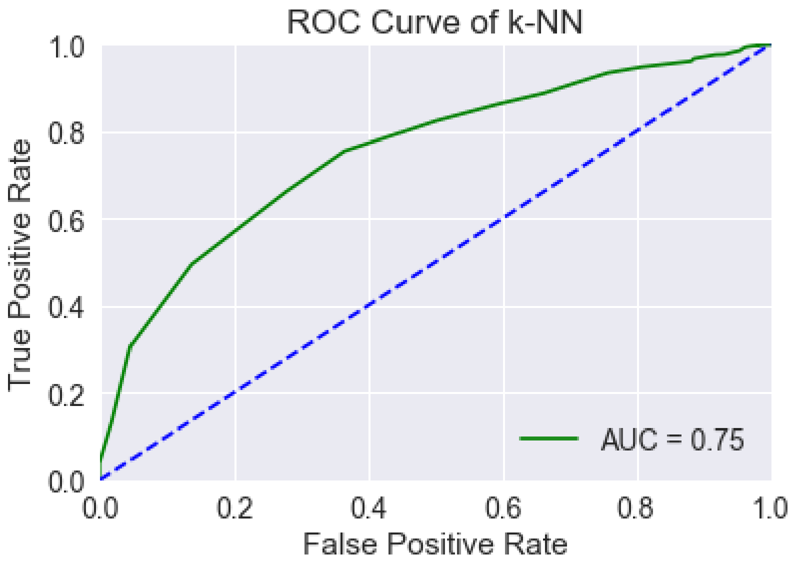
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
```

```
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.5646224205561694, 0.5770856352109504, 0.5796226172214415, 0.5833163486052
857, 0.5863289030011121, 0.5948373578199475, 0.6118846611814935, 0.650296964
5609178, 0.67908160893647, 0.6982904424253475, 0.7066558679553605, 0.7136982
725637643, 0.7210052813564898, 0.7254159470505572, 0.726533363369484, 0.7299
48187640124, 0.7241680165055799, 0.7278116876383363, 0.7302753671383058, 0.7
315483278088271, 0.728884407304506, 0.7362575671433118, 0.7481272102494789,
0.7536964131830095, 0.7499436822175262]
the optimal value of k using roc is 47
```

In [232]:

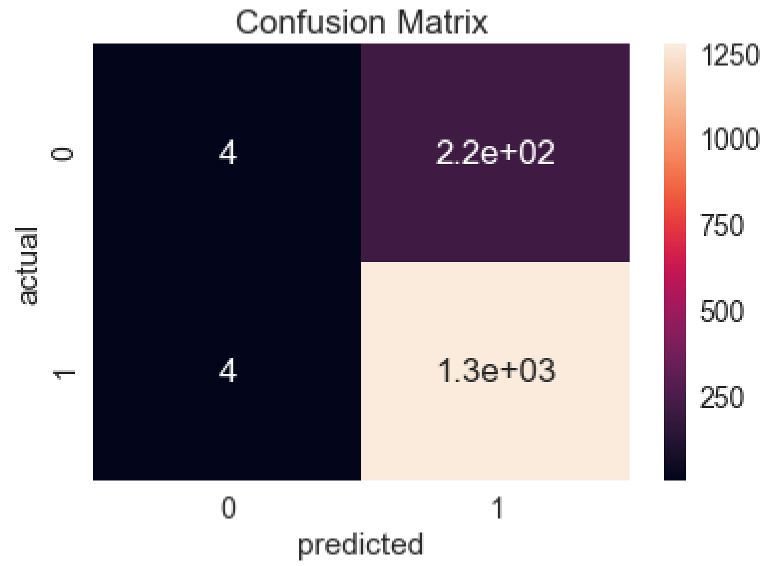
```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
plt.show()
predict=knn.predict(test_reviews)
```



In [233]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
print(cm)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```

```
[[   4  215]
 [   4 1273]]
```



In [234]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.50      0.02      0.04       219
          1       0.86      1.00      0.92      1277

avg / total       0.80      0.85      0.79      1496
```

[5.1.2] Applying KNN brute force on TFIDF, SET 2

In [236]:
```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, t
est_size=0.3, random_state=0)
print(len(x_train), len(x_test), len(y_train), len(y_test))
```

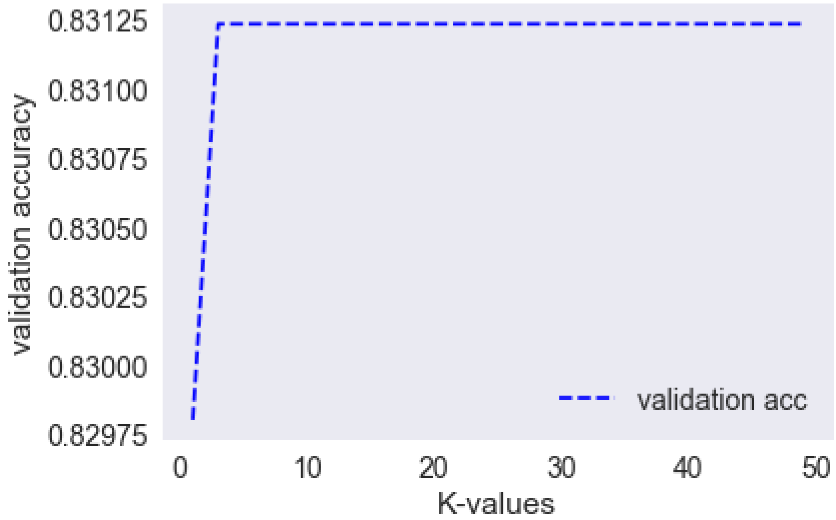
```
3490 1496 3490 1496
```

In [240]:
```
from sklearn.neighbors import KNeighborsClassifier
tf_idf_vect = TfidfVectorizer(ngram_range=(1,1))
train_reviews=tf_idf_vect.fit_transform(x_train)
test_reviews=tf_idf_vect.transform(x_test)
print(train_reviews.shape, test_reviews.shape)
```

```
(3490, 10976) (1496, 10976)
```

10-fold cross validation for TF-IDF

In [241]:
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 3
```

In [242]:
```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn_model.fit(train_reviews, y_train)
predict=knn_model.predict(test_reviews)
```

```
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=85.36096256684492
```

Area under roc curve for TF-IDF

In [243]:

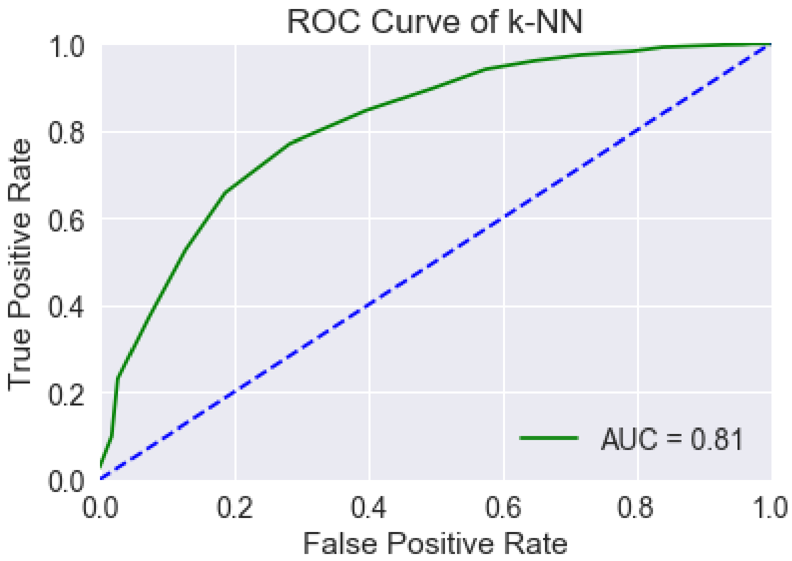
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.5082831836889399, 0.507891641010788, 0.507891641010788, 0.50789164101078
8, 0.507891641010788, 0.507891641010788, 0.6168030808508813, 0.6964614553945
284, 0.7214951566707073, 0.7406163847201811, 0.749355474267243, 0.7515617010
47332, 0.7648705763722767, 0.7676596475043177, 0.7795346542088155, 0.7848285
257613629, 0.7843350747149248, 0.7890192839238657, 0.795857871795697, 0.7962
976868588265, 0.8019741617589742, 0.8053746831007319, 0.8066941282901207, 0.
8091023839406715, 0.8121489077925932]
the optimal value of k using roc is 49
```

In [244]:

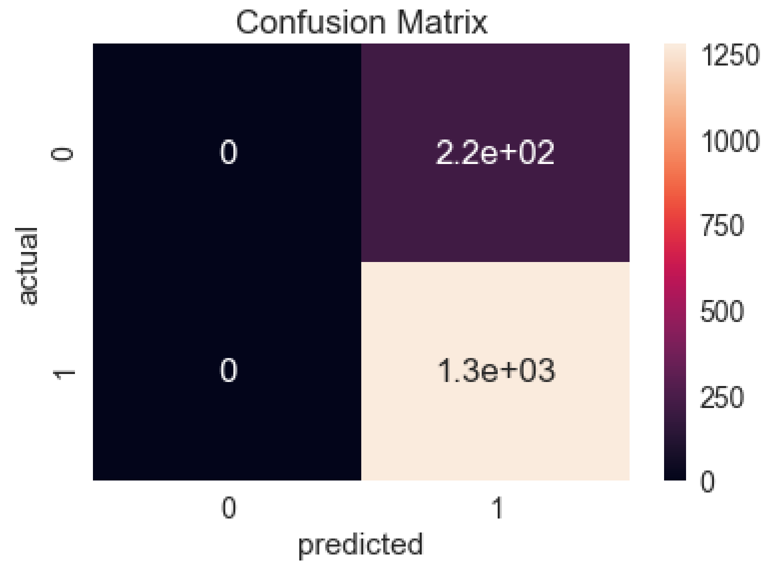
```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
```

```
plt.show()
predict=knn.predict(test_reviews)
```



In [245]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```



In [246]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       219
          1       0.85      1.00      0.92      1277

avg / total       0.73      0.85      0.79      1496
```

[5.1.3] Applying KNN brute force on AVG W2V, SET 3

In [261]:
```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, t
est_size=0.3, random_state=0)
```

In [361]:
```
import re

def cleanhtml(sentence):
    cleantext = re.sub('<.*>', '', sentence)
    return cleantext

def cleanpunc(sentence):
```

```
    cleaned = re.sub(r'[?|!|\'|#|@|.|,|)|(|\|/]', r'', sentence)
    return cleaned
```

In [362]:

```
train_sent_list = []
for sent in x_train:
    train_sentence = []
    sent = cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if (cleaned_words.isalpha()):
                train_sentence.append(cleaned_words.lower())
            else:
                continue
    train_sent_list.append(train_sentence)
```

In [363]:

```
test_sent_list = []
for sent in x_test:
    train_sentence = []
    sent = cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if (cleaned_words.isalpha()):
                train_sentence.append(cleaned_words.lower())
            else:
                continue
    test_sent_list.append(train_sentence)
```

In [364]:

```
import gensim
train_w2v_model = gensim.models.Word2Vec(train_sent_list, min_count=5, size=50
, workers=4)
train = train_w2v_model[train_w2v_model.wv.vocab]

test_w2v_model = gensim.models.Word2Vec(test_sent_list, min_count=5, size=50,
workers=4)
test = test_w2v_model[test_w2v_model.wv.vocab]
```

In [266]:

```
import numpy as np
train_vectors = []
for sent in train_sent_list:
    sent_vec = np.zeros(50)
    cnt_words = 0
    for word in sent:
        try:
```

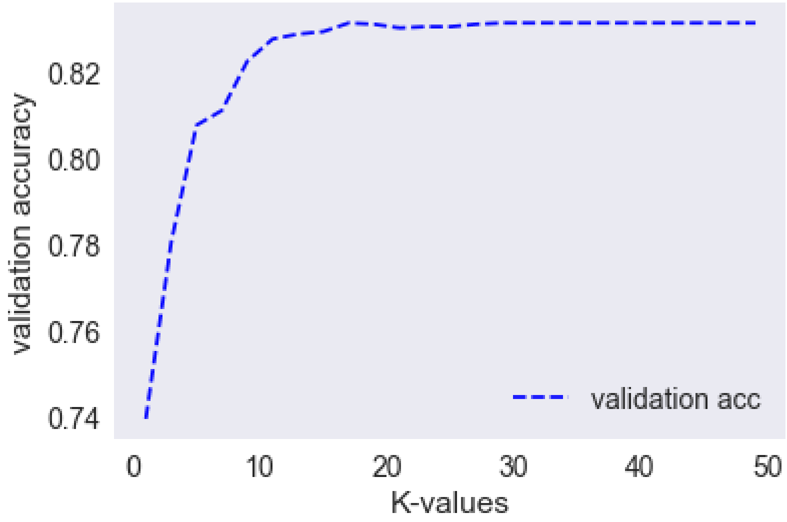
```
                vec = train_w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
            except:
                pass
        sent_vec /= cnt_words
        train_vectors.append(sent_vec)
train_reviews = np.nan_to_num(train_vectors)

test_vectors = []
for sent in test_sent_list:
    sent_vec = np.zeros(50)
    cnt_words = 0
    for word in sent:
        try:
            vec = test_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    test_vectors.append(sent_vec)
test_reviews = np.nan_to_num(test_vectors)
```

10-Fold cv for finding hyper parametre

In [267]:

```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 17
```

In [268]:
```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn_model.fit(train_reviews, y_train)
predict=knn_model.predict(test_reviews)
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=85.36096256684492
```

Finding Hyper parameter using area under roc curve

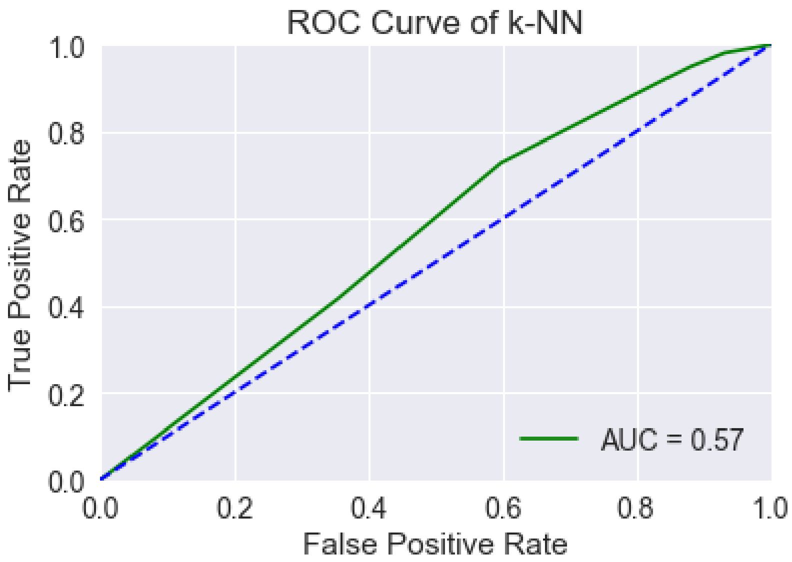
In [269]:
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.4988253719655442, 0.5383032435466973, 0.5465274276539979, 0.5655288686740
```

```
827, 0.5575317435627881, 0.5533570761952777, 0.5544798561125355, 0.555203941
8872, 0.5670342519389409, 0.5551324272427887, 0.551966116361478, 0.547453542
2991244, 0.5553737891676769, 0.5583917071618341, 0.5566842950265141, 0.55296
73213832362, 0.5536538619695849, 0.5570668983741146, 0.5597361824767667, 0.5
597415460750975, 0.547312300876412, 0.5350975996109604, 0.5286791602750454,
0.5219514201020514, 0.5253733958371325]
the optimal value of k using roc is 17
```

In [270]:

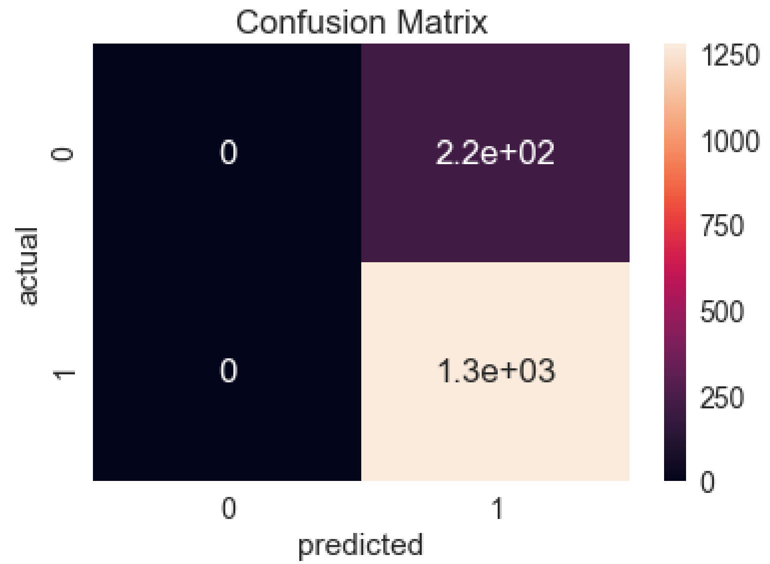
```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
plt.show()
predict=knn.predict(test_reviews)
```



In [271]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
```

```
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```



In [272]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       219
          1       0.85      1.00      0.92      1277

avg / total       0.73      0.85      0.79      1496
```

[5.1.4] Applying KNN brute force on TFIDF W2V, SET 4

In [316]:
```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, t
est_size=0.3, random_state=0)
```

In [317]:
```
tfidf_vect = TfidfVectorizer(ngram_range=(1, 1))
train_tfidf_w2v = tfidf_vect.fit_transform(x_train)
test_tfidf_w2v = tfidf_vect.transform(x_test)
print(train_tfidf_w2v.shape, test_tfidf_w2v.shape)
```

```
(3490, 10976) (1496, 10976)
```

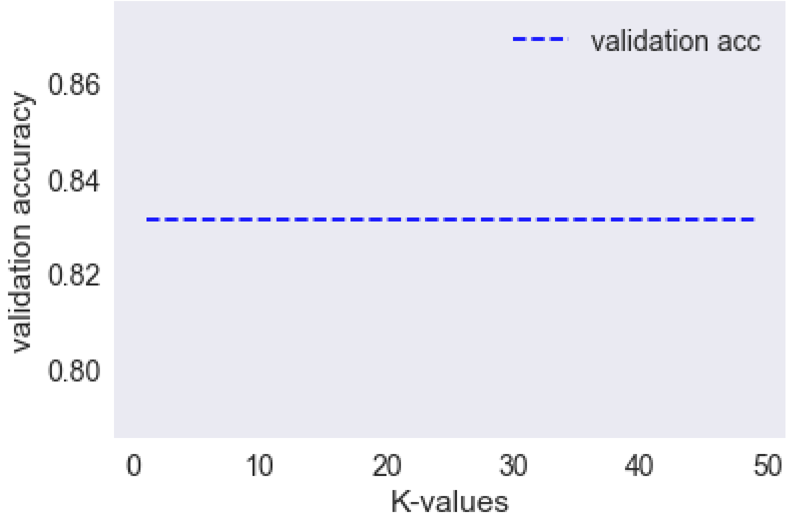
In [318]:
```
tfidf_feat = tfidf_vect.get_feature_names()
train_reviews = []
row = 0
for sent in train_sent_list:
    sent_vec = np.zeros(50)
    weight_sum = 0
    for word in sent:
        if word in train_w2v_words:
            vec = train_w2v_model.wv[word]
            tf_idf = train_tfidf_w2v[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_reviews.append(sent_vec)
    row += 1
```

In [319]:
```
tfidf_feat = tfidf_vect.get_feature_names()
test_reviews = []
row = 0
for sent in test_sent_list:
    sent_vec = np.zeros(50)
    weighted_sum = 0
    for word in sent:
        if word in test_w2v_words:
            vec = test_w2v_model[word]
            tf_idf = test_tfidf_w2v[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    test_reviews.append(sent_vec)
    row += 1
```

10-Fold cv for finding Hyper parameter

In [321]:

```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 1
```

In [311]:

```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn_model.fit(train_reviews, y_train)
predict=knn_model.predict(test_reviews)
```

```
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=85.36096256684492
```

Hyper parameter using area underroc curve

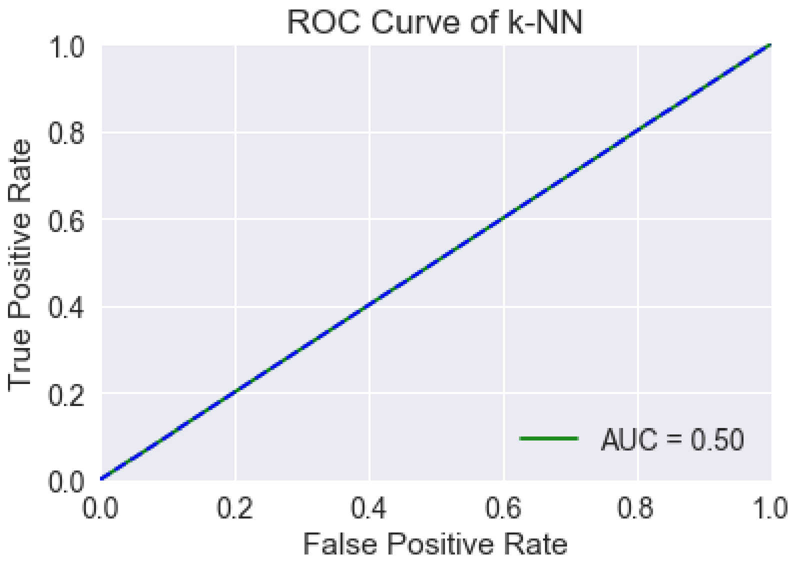
In [312]:

```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='brute')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

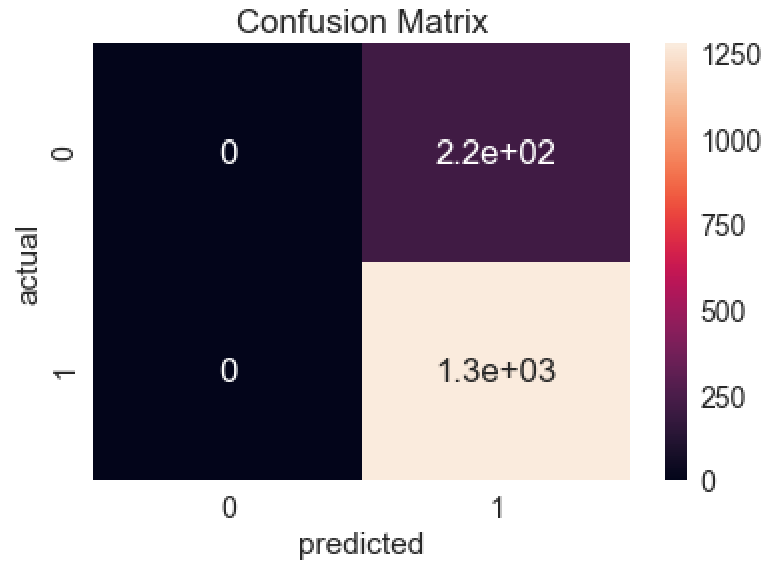
```
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
the optimal value of k using roc is 1
```

In [313]:

```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='brute')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
plt.show()
predict=knn.predict(test_reviews)
```



In [314]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```



In [315]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       219
          1       0.85      1.00      0.92      1277

avg / total       0.73      0.85      0.79      1496
```

[5.2] Applying KNN kd-tree

[5.2.1] Applying KNN kd-tree on BOW, SET 5

In [322]:
```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, t
est_size=0.3, random_state=0)
print(len(x_train), len(x_test), len(y_train), len(y_test))
```

```
3490 1496 3490 1496
```

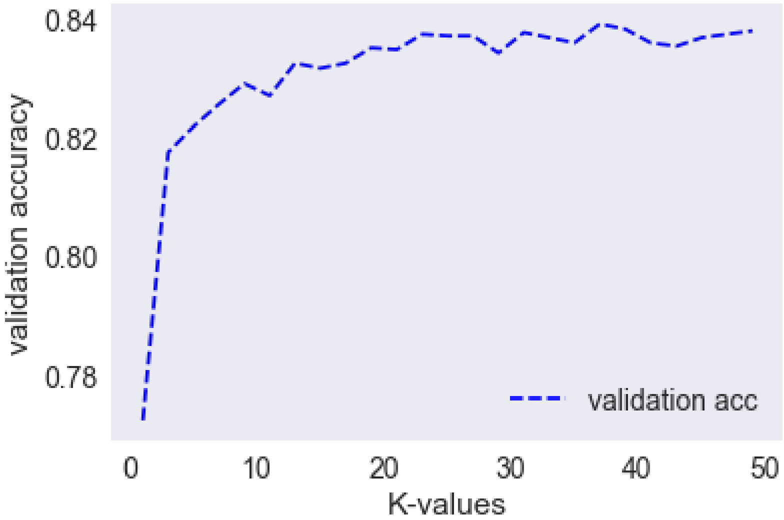
In [337]:
```
from sklearn.neighbors import KNeighborsClassifier
count_vect = CountVectorizer(ngram_range=(1,1), max_features=500)
```

```
train_reviews=count_vect.fit_transform(x_train).toarray()
test_reviews=count_vect.transform(x_test).toarray()
```

using 10-fold cv to get Hyper parameter

In [338]:

```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 37
```

In [339]:

```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn_model.fit(train_reviews, y_train)
```

```
predict=knn_model.predict(test_reviews)
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=85.09358288770053
```

Hyper parameter using area under roc curve

In [340]:

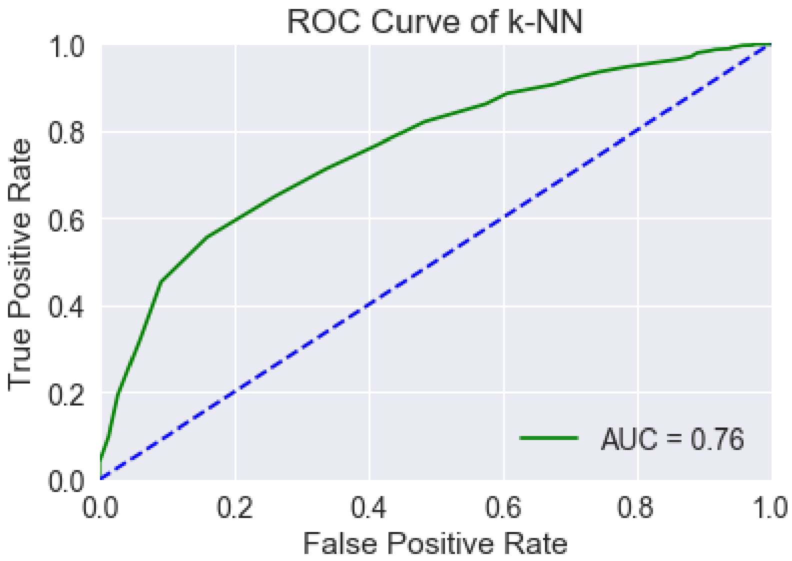
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.615043820598363, 0.6530931871574002, 0.6696863725269342, 0.67767634617378
77, 0.6751518792260686, 0.6744850051669332, 0.6775422562155166, 0.6977969913
789097, 0.7090837901331246, 0.7099902382510379, 0.7169754311439125, 0.727087
6018636717, 0.735411906473148, 0.7391324558486465, 0.7438738767731162, 0.737
8773738392279, 0.7416926801185713, 0.7456992880717149, 0.7489138713380032,
0.744555053761134, 0.7494555947694189, 0.7522178479098058, 0.754649345819790
3, 0.7590993445682841, 0.760667303147002]
the optimal value of k using roc is 49
```

In [341]:

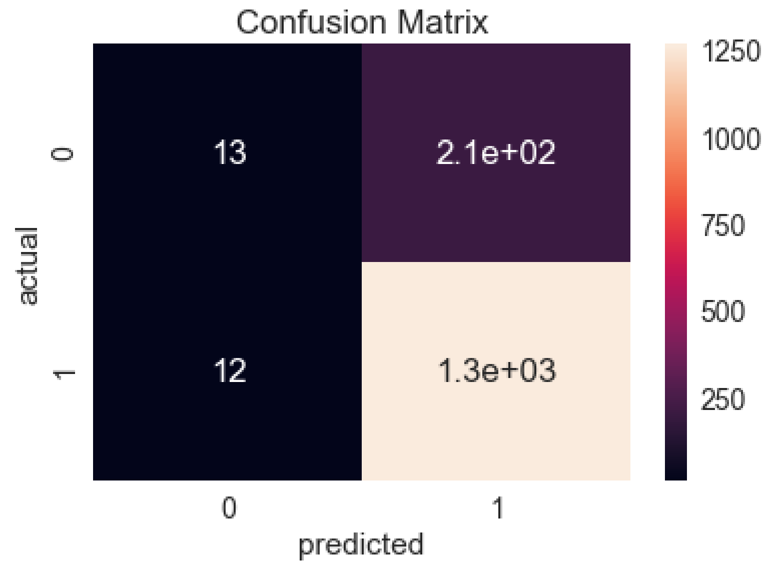
```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
```

```
plt.show()
predict=knn.predict(test_reviews)
```



In [342]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
print(cm)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```

```
[[  13  206]
 [  12 1265]]
```



In [343]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.52      0.06      0.11       219
          1       0.86      0.99      0.92      1277

avg / total       0.81      0.85      0.80      1496
```

[5.2.2] Applying KNN kd-tree on TFIDF, SET 6

In [344]:
```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, t
est_size=0.3, random_state=0)
print(len(x_train), len(x_test), len(y_train), len(y_test))
```

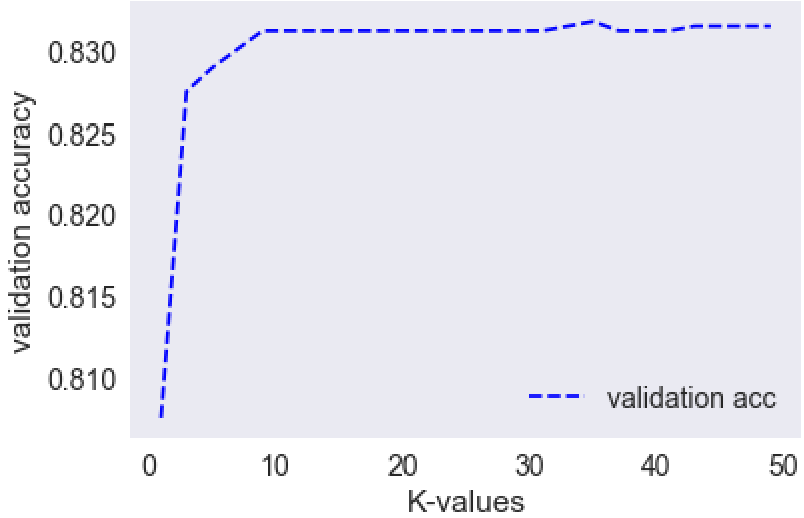
```
3490 1496 3490 1496
```

In [345]:
```
from sklearn.neighbors import KNeighborsClassifier
tf_idf_vect = TfidfVectorizer(ngram_range=(1,1), max_)
train_reviews=tf_idf_vect.fit_transform(x_train).toarray()
test_reviews=tf_idf_vect.transform(x_test).toarray()
print(train_reviews.shape, test_reviews.shape)
```

```
(3490, 500) (1496, 500)
```

10-fold cv to get Hyper parameter

In [346]:
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 35
```

In [347]:
```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn_model.fit(train_reviews, y_train)
predict=knn_model.predict(test_reviews)
```

```
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=85.36096256684492
```

Hyper parameter using area underroc curve

In [348]:

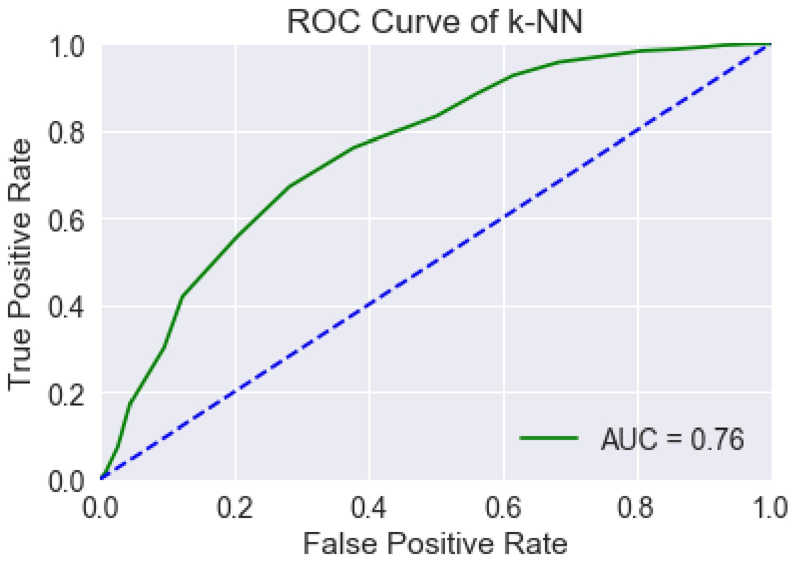
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.5274526841233914, 0.5360809259716158, 0.535512384548546, 0.53220483224452
29, 0.532591011324344, 0.531820441030812, 0.5842996749659412, 0.633004723542
2633, 0.6639115649907209, 0.6827628252575422, 0.7010151503774186, 0.71071253
61595922, 0.7248581328241491, 0.7281764123248338, 0.7258217926575913, 0.7305
310319920763, 0.7342462177692437, 0.7365758073109422, 0.7384459152622979, 0.
7428637324208065, 0.739464998945159, 0.7420287989473043, 0.7464108587836074,
0.7495306851460507, 0.7554270675777633]
the optimal value of k using roc is 49
```

In [349]:

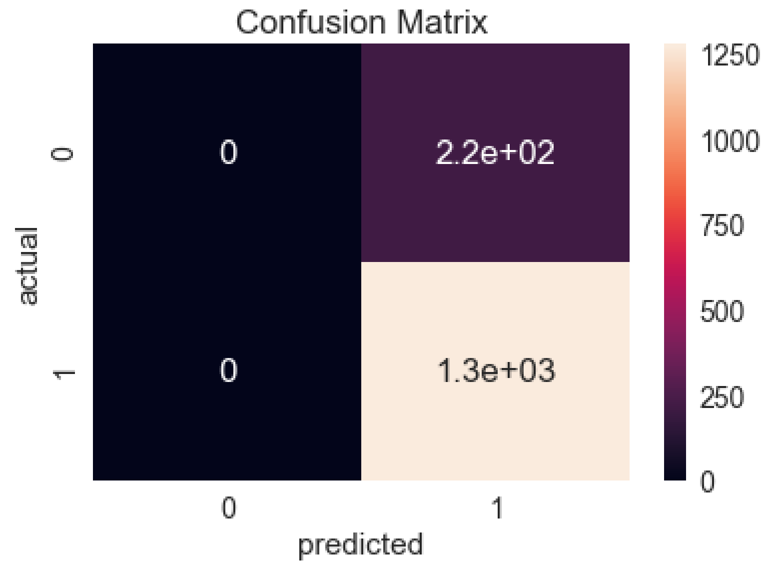
```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
```

```
plt.show()
predict=knn.predict(test_reviews)
```



In [350]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```



In [351]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       219
          1       0.85      1.00      0.92      1277

avg / total       0.73      0.85      0.79      1496
```

[5.2.3] Applying KNN kd-tree on AVG W2V, SET 7

In [369]:
```
import numpy as np
train_vectors = []
for sent in train_sent_list:
    sent_vec = np.zeros(50)
    cnt_words = 0
    for word in sent:
        try:
            vec = train_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
```

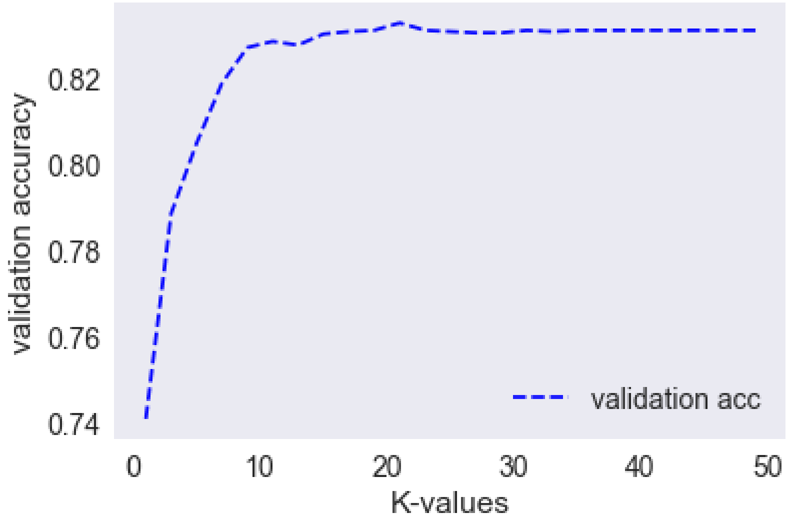
```
    train_vectors.append(sent_vec)
train_reviews = np.nan_to_num(train_vectors)

test_vectors = []
for sent in test_sent_list:
    sent_vec = np.zeros(50)
    cnt_words = 0
    for word in sent:
        try:
            vec = test_w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    test_vectors.append(sent_vec)
test_reviews = np.nan_to_num(test_vectors)
```

10-fold cv to get Hyper parameter

In [370]:

```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 21
```

In [371]:
```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn_model.fit(train_reviews, y_train)
predict=knn_model.predict(test_reviews)
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=85.36096256684492
```

Hyper parameter using roc curve

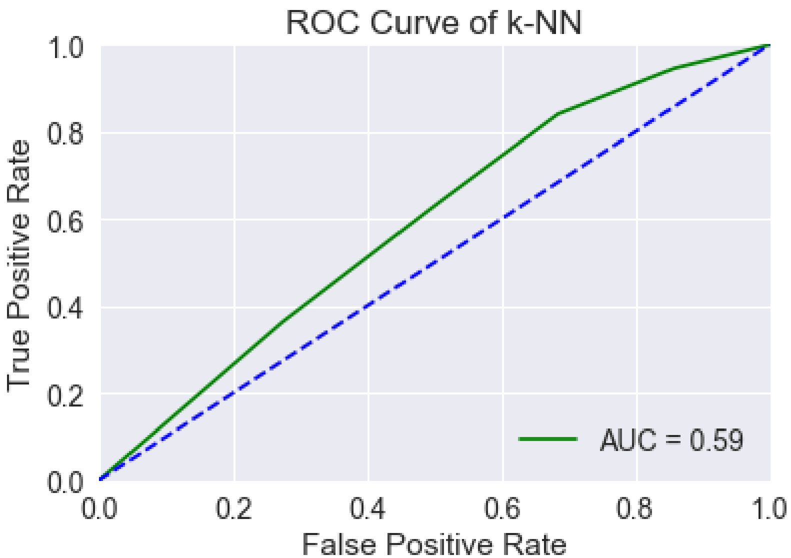
In [372]:
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.5, 0.5344897251334642, 0.5359700782727783, 0.5486728669863372, 0.57925252
```

```
89366129, 0.5758859770509506, 0.5826441109478193, 0.5811834243357183, 0.5745
629561293414, 0.5688596632375396, 0.5731916628227545, 0.5832323188981023, 0.
5908164469379218, 0.5827263527888924, 0.5829265937932441, 0.583579164923497
3, 0.5811083339590866, 0.5793830431626636, 0.5749688017363755, 0.56722019001
44102, 0.5677136410608482, 0.5684466661660642, 0.5685521502665708, 0.5650675
992176297, 0.5663173176287174]
the optimal value of k using roc is 25
```

In [373]:

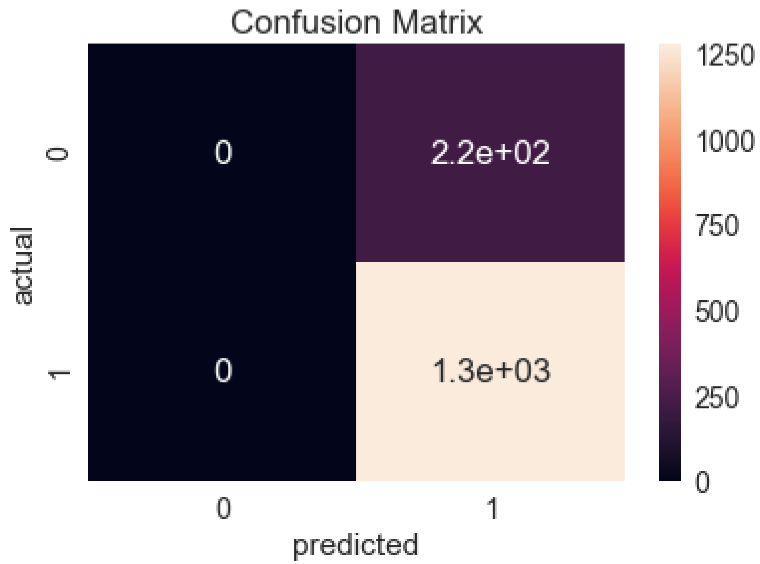
```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
plt.show()
predict=knn.predict(test_reviews)
```



In [374]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
```

```
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```



In [375]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       219
          1       0.85      1.00      0.92      1277

avg / total       0.73      0.85      0.79      1496
```

[5.2.4] Applying KNN kd-tree on TFIDF W2V, SET 8

In [376]:

```
from sklearn.cross_validation import train_test_split
x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews, labels, test_size=0.3, random_state=0)
print(len(x_train), len(x_test), len(y_train), len(y_test))
```

```
3490 1496 3490 1496
```

In [377]:

```
tfidf_vect = TfidfVectorizer(ngram_range=(1, 1))
train_tfidf_w2v = tfidf_vect.fit_transform(x_train)
test_tfidf_w2v = tfidf_vect.transform(x_test)
print(train_tfidf_w2v.shape, test_tfidf_w2v.shape)
```

```
(3490, 10976) (1496, 10976)
```

In [378]:

```
tfidf_feat = tfidf_vect.get_feature_names()
train_reviews = []
row = 0
for sent in train_sent_list:
    sent_vec = np.zeros(50)
    weight_sum = 0
    for word in sent:
        if word in train_w2v_words:
            vec = train_w2v_model.wv[word]
            tf_idf = train_tfidf_w2v[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    train_reviews.append(sent_vec)
    row += 1
```

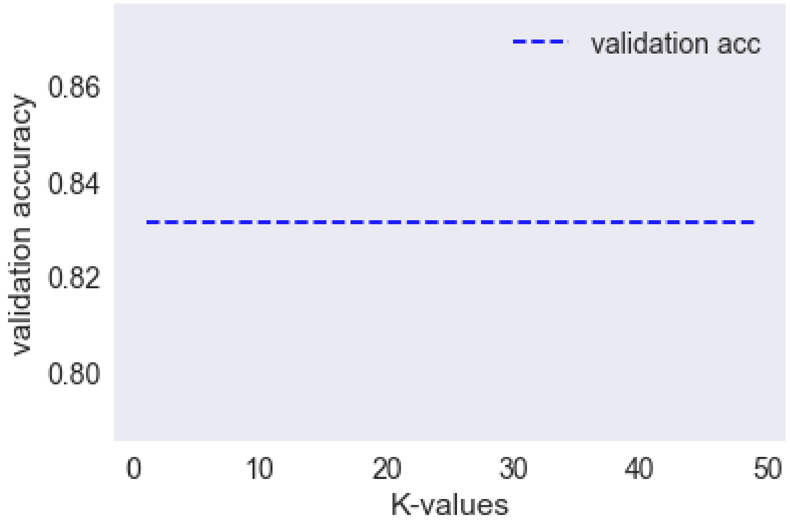
In [379]:

```
tfidf_feat = tfidf_vect.get_feature_names()
test_reviews = []
row = 0
for sent in test_sent_list:
    sent_vec = np.zeros(50)
    weighted_sum = 0
    for word in sent:
        if word in test_w2v_words:
            vec = test_w2v_model[word]
            tf_idf = test_tfidf_w2v[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
```

```
    test_reviews.append(sent_vec)
    row += 1
```

10-fold cv for Hyper parameter

In [380]:
```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
cv_scores=[]
for i in k_values:
    knn_model=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    scores=cross_val_score(knn_model, train_reviews, y_train,cv=10, scoring='a
ccuracy')
    cv_scores.append(scores.mean())
plt.plot(k_values, cv_scores, 'b--', label='validation acc')
plt.grid()
plt.xlabel('K-values')
plt.ylabel('validation accuracy')
plt.legend()
plt.show()
mse=[1-x for x in cv_scores]
best_k=k_values[mse.index(min(mse))]
print('the best k value using 10-fold cv is '+str(best_k))
```




```
the best k value using 10-fold cv is 1
```

In [381]:
```
from sklearn.metrics import accuracy_score
knn_model=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn_model.fit(train_reviews, y_train)
```

```
predict=knn_model.predict(test_reviews)
acc=accuracy_score(y_test, predict)*100
print('test accuracy='+str(acc))
```

```
test accuracy=14.63903743315508
```

Hyper paarameter using roc curve

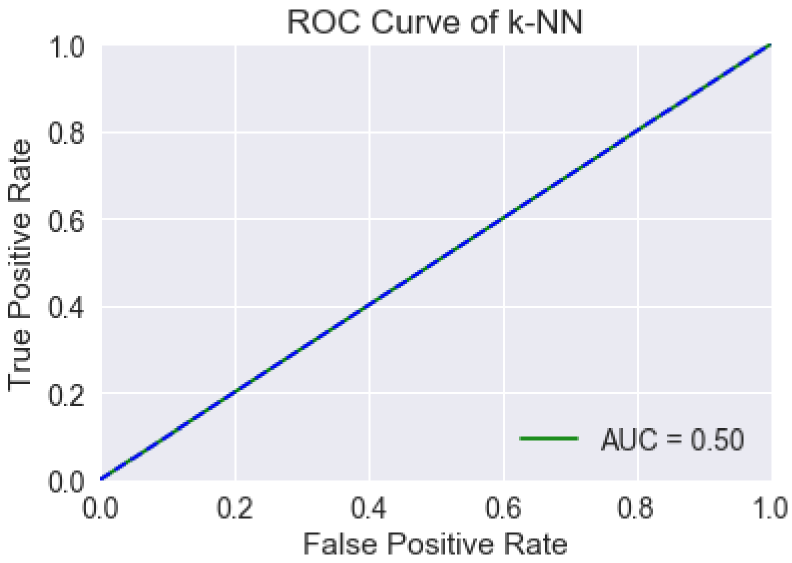
In [382]:

```
list1=list(range(1,50))
k_values=list(filter(lambda x:x%2!=0, list1))
aucs=[]
for i in k_values:
    knn=KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    knn.fit(train_reviews, y_train)
    prob=knn.predict_proba(test_reviews)
    fpr, tpr, thre=roc_curve(y_test,prob[:,1])
    roc_auc=auc(fpr, tpr)
    aucs.append(roc_auc)
print(aucs)
best_k=k_values[aucs.index(max(aucs))]
print('the optimal value of k using roc is '+str(best_k))
```

```
[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
the optimal value of k using roc is 1
```

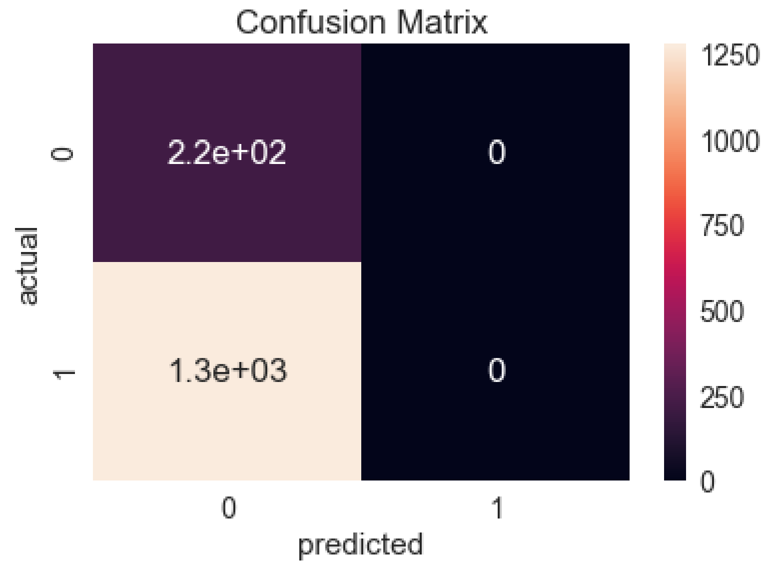
In [383]:

```
knn=KNeighborsClassifier(n_neighbors=best_k, algorithm='kd_tree')
knn.fit(train_reviews, y_train)
prob=knn.predict_proba(test_reviews)
fpr, tpr, thre=roc_curve(y_test,prob[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'g', label = 'AUC = %0.2f' % max(aucs))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'b--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of k-NN')
plt.show()
predict=knn.predict(test_reviews)
```



In [384]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
#from mlxtend.plotting import plot_confusion_matrix
lab=[0,1]
cm=confusion_matrix(y_test, predict, labels=lab)
df_cm=pd.DataFrame(cm)
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True)
#plt.matshow(b)
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('Confusion Matrix')
#fig, ax = plot_confusion_matrix(conf_mat=binary,show_absolute=True,show_norme
d=True,colorbar=True)
#plt.colorbar()
plt.show()
```



In [385]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict))
```

```
             precision    recall  f1-score   support

          0       0.15      1.00      0.26       219
          1       0.00      0.00      0.00      1277

avg / total       0.02      0.15      0.04      1496
```

[6] Conclusions

In [397]:
```
data=[['BOW', 'Brute',21, 47, 0.75],['TF-IDF', 'Brute',3,49,0.81],['Avg w2v', 'Brute',17,17,0.57],['TF-IDF w2v', 'Brute',1,1,0.5],['BOW', 'kd tree',37,49, 0.76],['TF-IDF', 'kd tree', 35, 49, 0.76],['Avg w2v','kd tree',21,25,0.59], [ 'TF-IDF w2v', 'kd tree',1,1,0.5]]
```

In [400]:
```
new_df=pd.DataFrame(data=data, columns=['vectorizer', 'model', 'hyper parameter(10 fold-cv)', 'hyper parameter(roc)', 'AUC'])
```

In [401]:
```
new_df
```

Out[401]:

	vectorizer	model	hyper parameter(10 fold-cv)	hyper parameter(roc)	AUC

0	BOW	Brute	21	47	0.75
1	TF-IDF	Brute	3	49	0.81
2	Avg w2v	Brute	17	17	0.57
3	TF-IDF w2v	Brute	1	1	0.50
4	BOW	kd tree	37	49	0.76
5	TF-IDF	kd tree	35	49	0.76
6	Avg w2v	kd tree	21	25	0.59
7	TF-IDF w2v	kd tree	1	1	0.50