Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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 [2]:

```
%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

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
import os
```

In [3]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/AI/amazon-fine-food-reviews/database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 30000""", 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(5)
```

```
Number of data points in our data (30000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	0	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	1	1350777600	Great taffy	Great taffy at a great price. There was a wid...

```
In [4]: print(filtered_data.shape)
```

```
(30000, 10)
```

[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 [5]:

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

Out[5]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Te
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOU WAFERS FIND THA EUROPEA WAFERS
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOU WAFERS FIND THA EUROPEA WAFERS
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOU WAFERS FIND THA EUROPEA WAFERS
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOU WAFERS FIND THA EUROPEA WAFERS
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577600	LOACKER QUADRATINI VANILLA WAFERS	DELICIOU WAFERS FIND THA EUROPEA WAFERS

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 [6]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='l
```

In [7]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[7]:
```
(28072, 10)
```

In [8]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[8]:
```
93.57333333333332
```

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 [9]:

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

display.head()
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892800	Bought This for My Son at College	My son loves spaghetti so I didn't hesitate or...
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883200	Pure cocoa taste with crunchy almonds inside	It was almost a 'love at first bite' - the per...

In [10]:

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

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

```
(28072, 10)
```

Out[10]:

```
1    23606
0     4466
Name: Score, dtype: int64
```

In [11]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

[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

Preprocessing Review

In [12]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [13]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe positive reviews
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative reviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
```

In [14]:
```
final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-processing of the revi
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
print(final.shape)
final.head(5)
```

```
(28072, 11)
```

Out[14]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Tex
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0	1	1195948800	Dog Lover Delites	Our dog: just love them. saw then in a pet ..
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1	0	1192060800	made in china	My dog: loves thi: chicker but its a produc f..
2546	2774	B00002NCJC	A196AJHU9EASJN	Alex Chaffee	0	0	1	1282953600	thirty bucks?	Why i: this $[... when the same product i: av..
2547	2775	B00002NCJC	A13RRPGE79XFFH	reader48	0	0	1	1281052800	Flies Begone	We have used the Victor fl bait for : seasons..
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	962236800	WOW Make your own 'slickers' !	I jus receivec my shipmen and coulc hardly w..

[4] Featurization

In [16]:
```
#storing the cleaned data in the form of sql table for furthur use
conn = sqlite3.connect('final.sqlite')
c=conn.cursor()
conn.text_factory = str
final.to_sql('Reviews',conn,schema=None,if_exists='replace',index=True,index_label=None,chunksize=None,dtype=None)
```

In [17]:
```
#reading or extracting the newly stored sql table and we will be using same throughtout our analysis
con=sqlite3.connect('finals.sqlite')
cleaned_data=pd.read_sql_query('select * from Reviews',con)
cleaned_data.shape
cleaned_data['Score'].size
```

Out[17]:
```
28072
```

In [18]:

```
#Taking Sample Data
n_samples = 25000
cleaned_data_sample = cleaned_data.sample(n_samples)

###Sorting as we want according to time series
cleaned_data_sample.sort_values('Time',inplace=True)
cleaned_data_sample.head(5)
```

Out[18]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	
5	1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7	1	961718400	Great Product	Thi a goo ai r
4	1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	962236800	WOW Make your own 'slickers' !	rec ship and I
266	28087	30630	B00008RCMI	A284C7M23F0APC	A. Mendoza	0	0	1	1067040000	Best sugarless gum ever!	I lov stu s fre c
265	28086	30629	B00008RCMI	A19E94CF5O1LY7	Andrew Arnold	0	0	1	1067040000	I've chewed this gum many times, but used?	N a pr bo
405	10992	11991	B0000T15M8	A2928LJN5IISB4	chatchi	5	5	1	1067990400	The fruits of my labor	cha fa afte beca

In [19]:

```
import pickle
def savetofile(obj,filename):
    pickle.dump(obj,open(filename+".p","wb"))
def openfromfile(filename):
    temp = pickle.load(open(filename+".p","rb"))
    return temp
#Saving 20000 samples in disk to as to test to test on the same sample for each of all Algo
savetofile(cleaned_data_sample,"sample_25000")
```

In [20]:

```
#Opening from samples from file
cleaned_data_sample = openfromfile("sample_25000")
```

BOW with Brute Force

In [19]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(cleaned_data_sample['CleanedText'].values,cleaned_data_sample['Score']

#BoW
count_vect = CountVectorizer() #in scikit-learn

X_train_bow = count_vect.fit_transform(X_train)
X_train_bow = preprocessing.normalize(X_train_bow)

X_test_bow = count_vect.transform(X_test)
X_test_bow = preprocessing.normalize(X_test_bow)

print("the type of count vectorizer for train data",type(X_train_bow))
print("the type of count vectorizer for test data",type(X_test_bow))

print("the shape of out train data BOW vectorizer ",X_train_bow.get_shape())
print("the number of unique words for train data", X_train_bow.get_shape()[1])

print("the shape of out test data BOW vectorizer ",X_test_bow.get_shape())
print("the number of unique words for test data", X_test_bow.get_shape()[1])
```

```
the type of count vectorizer for train data <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer for test data <class 'scipy.sparse.csr.csr_matrix'>
the shape of out train data BOW vectorizer  (17500, 15842)
the number of unique words for train data 15842
the shape of out test data BOW vectorizer  (7500, 15842)
the number of unique words for test data 15842
```

In [20]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

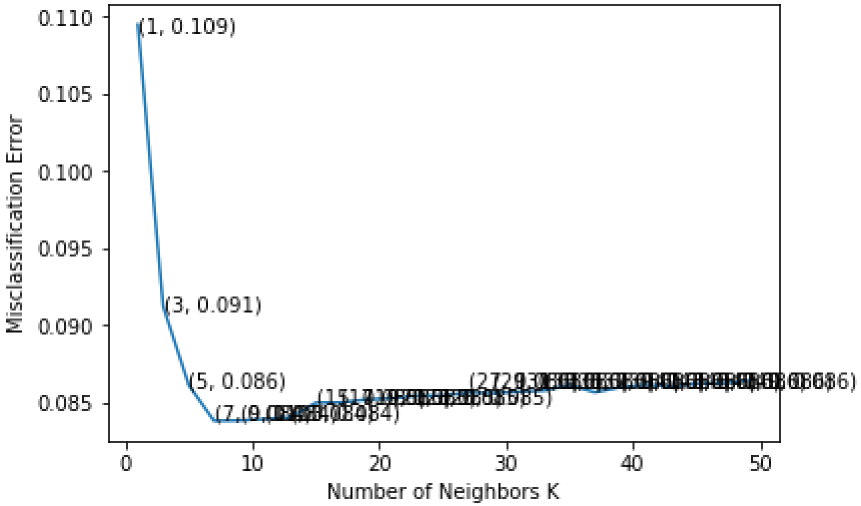
In [21]:

```
#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')

    score = cross_val_score(knn, X_train_bow, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

# print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [22]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 7.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.109 0.091 0.086 0.084 0.084 0.084 0.084 0.085 0.085 0.085 0.085
0.085
 0.085 0.086 0.086 0.086 0.086 0.086 0.086 0.086 0.086 0.086 0.086 0.086
 0.086]
```

In [23]:

```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_bow, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_bow)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 7 is 0.799893
```

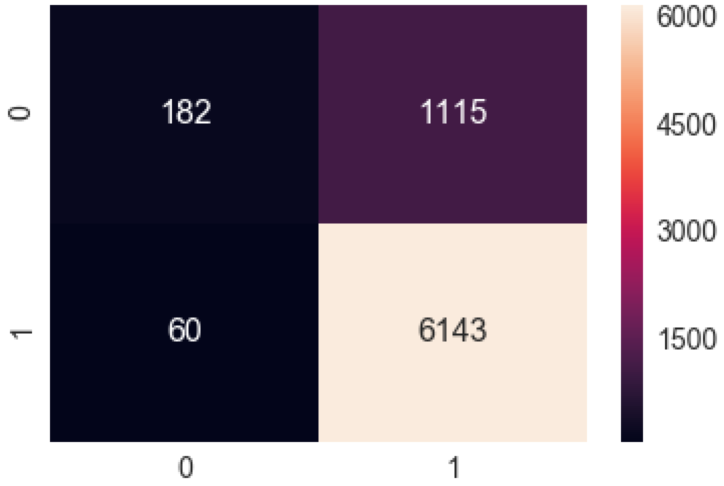
In [24]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='brute')
knn.fit(X_train_bow,y_train)
y_pred = knn.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 84.333%
Precision on test set: 0.846
Recall on test set: 0.990
F1-Score on test set: 0.913
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[24]:

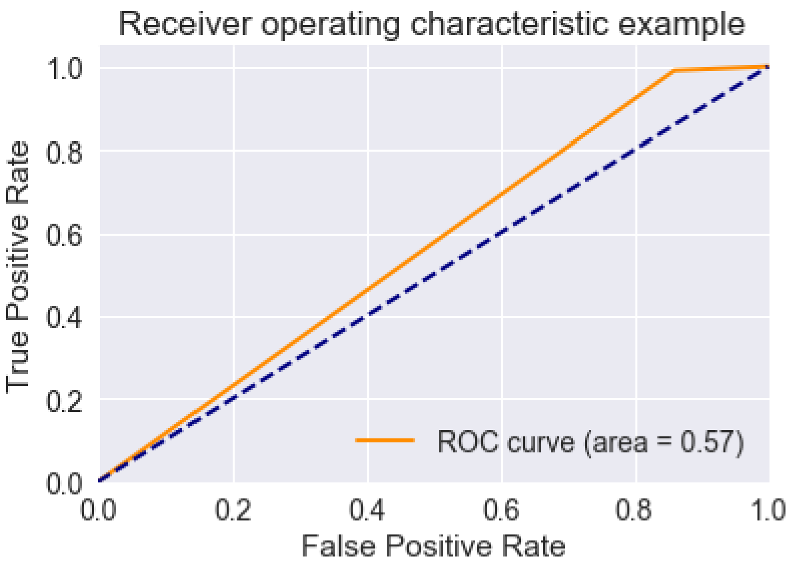
```
<matplotlib.axes._subplots.AxesSubplot at 0xc398e57668>
```


0
1
182
1115
60
6143
0
1
6000
4500
3000
1500

In [25]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.5653255426062276
```



[4.3] TF-IDF with Brute force

In [26]:

```
#Taking Sample Data
n_samples = 25000
cleaned_data_sample = cleaned_data.sample(n_samples)

###Sorting as we want according to time series
cleaned_data_sample.sort_values('Time',inplace=True)
```

In [27]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(cleaned_data_sample['CleanedText'].values,cleaned_data_sample['Score']

tfidf = TfidfVectorizer(ngram_range=(1,2)) #Using bi-grams
X_train_tfidf = tfidf.fit_transform(X_train)
#Normalize Data
X_train = preprocessing.normalize(X_train_tfidf)
print("Train Data Size: ",X_train_tfidf.shape)
X_test_tfidf = tfidf.transform(X_test)
#Normalize Data
X_test_tfidf = preprocessing.normalize(X_test_tfidf)
print("Test Data Size: ",X_test_tfidf.shape)
```

```
Train Data Size:  (17500, 347802)
Test Data Size:  (7500, 347802)
```

In [28]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

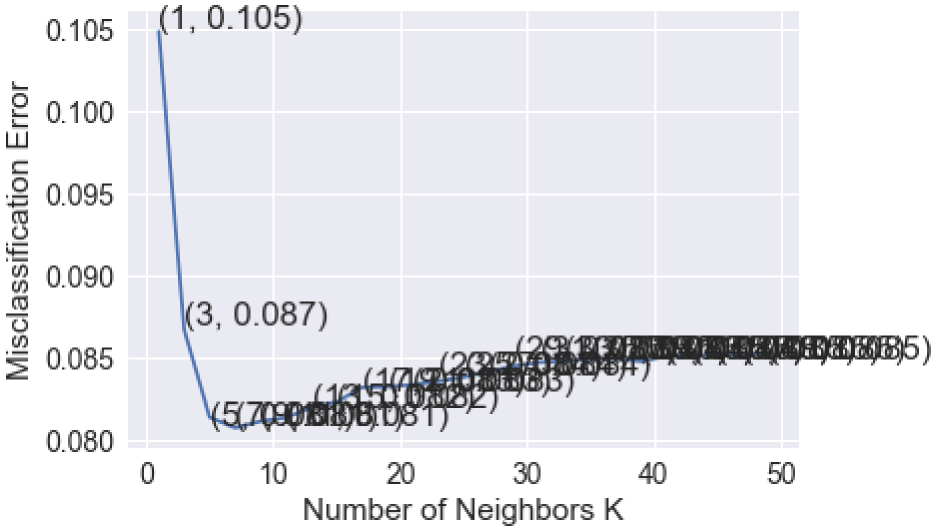
In [29]:

```

#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')

    score = cross_val_score(knn, X_train_tfidf, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

# print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [30]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 7.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.105 0.087 0.081 0.081 0.081 0.081 0.082 0.082 0.083 0.083 0.083
0.084
 0.084 0.084 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085
 0.085]
```

In [31]:

```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_tfidf, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_tfidf)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```

The accuracy of the knn classifier for k = 7 is 0.806968
```

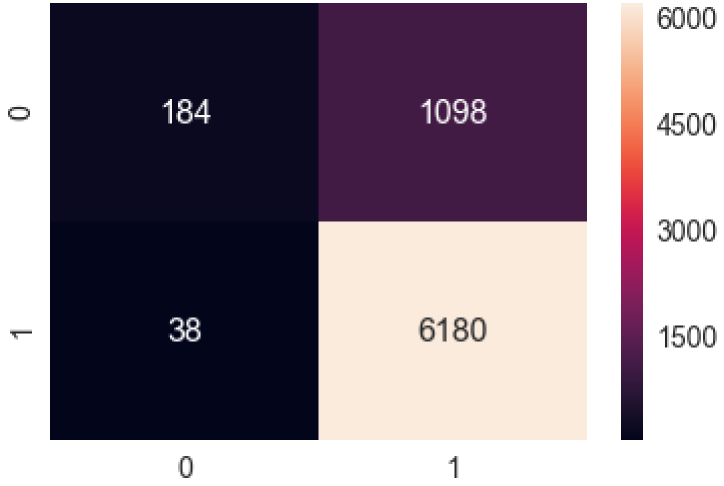
In [32]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='brute')
knn.fit(X_train_tfidf,y_train)
y_pred = knn.predict(X_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 84.853%
Precision on test set: 0.849
Recall on test set: 0.994
F1-Score on test set: 0.916
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[32]:

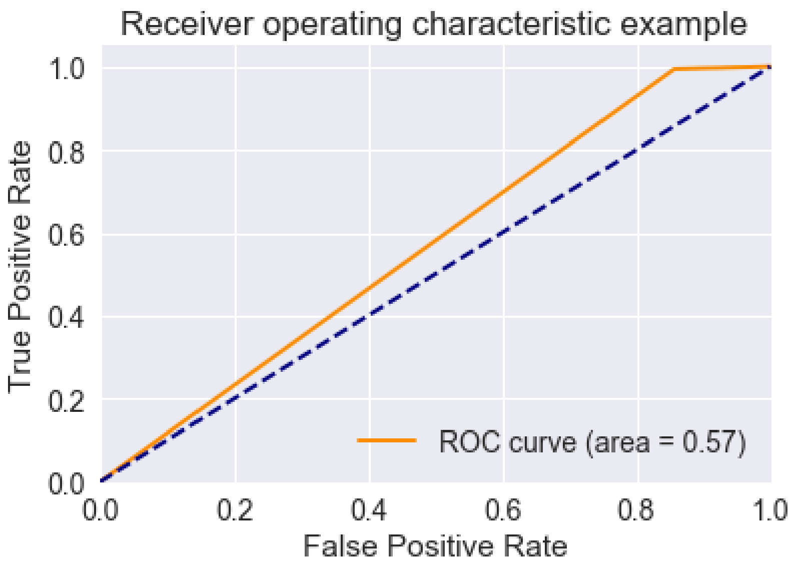
```
<matplotlib.axes._subplots.AxesSubplot at 0xc395fbf3c8>
```


184
1098
38
6180
0
1
0
1
6000
4500
3000
1500

In [33]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.5687072256129229
```



[4.4] Word2Vec with Brute force

In [34]:
```
#Taking Sample Data
n_samples = 25000
cleaned_data_sample = cleaned_data.sample(n_samples)

###Sorting as we want according to time series
cleaned_data_sample.sort_values('Time',inplace=True)
```

In [35]:
```
i=0
list_of_sent=[]
for sent in cleaned_data_sample['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [36]:
```
print(cleaned_data_sample['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
realli good idea final product outstand use decal car window everybodi ask bought decal made two thumb
*****************************************************************
['realli', 'good', 'idea', 'final', 'product', 'outstand', 'use', 'decal', 'car', 'window', 'everybodi', 'ask', 'bough
t', 'decal', 'made', 'two', 'thumb']
```

In [37]:
```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [38]:
```
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  6506
sample words  ['realli', 'good', 'idea', 'final', 'product', 'outstand', 'use', 'car', 'window', 'everybodi', 'ask',
'bought', 'made', 'two', 'thumb', 'receiv', 'shipment', 'could', 'hard', 'wait', 'tri', 'love', 'call', 'instead', 'st
icker', 'remov', 'easili', 'daughter', 'design', 'sign', 'print', 'revers', 'beauti', 'shop', 'program', 'go', 'lot',
'fun', 'everywher', 'surfac', 'like', 'screen', 'comput', 'monitor', 'noth', 'bother', 'link', 'top', 'page', 'buy']
```

In [39]:

```
# Average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    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]))
sent_vectors = np.array(sent_vectors)
```

```
25000
50
```

In [40]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

avg_vec_norm = preprocessing.normalize(sent_vectors)

#Not shuffling the data as we want it on time basis
X_train_w2v, X_test_w2v, y_train, y_test = train_test_split(avg_vec_norm,cleaned_data_sample['Score'].values,test_size=0.3
```

In [41]:

```
avg_vec_norm.shape
```

Out[41]:

```
(25000, 50)
```

In [42]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

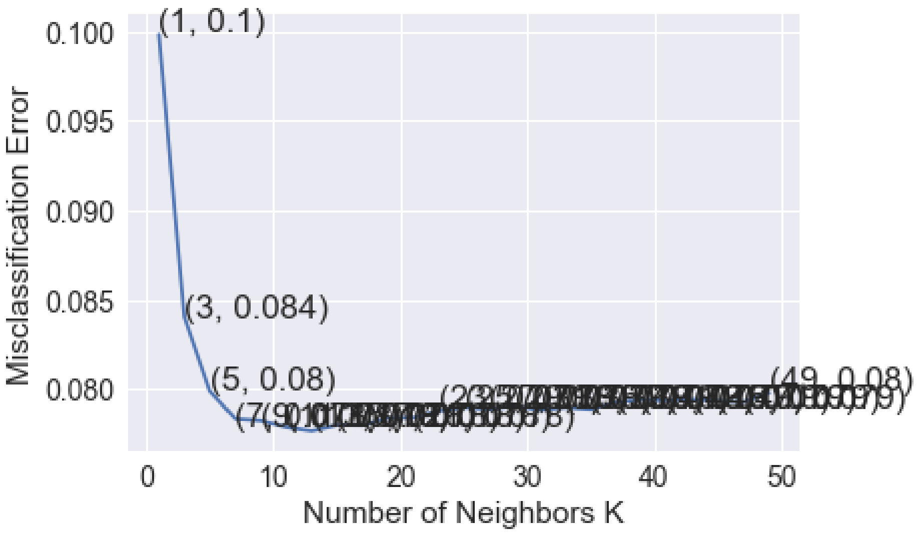
In [43]:

```
#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')

    score = cross_val_score(knn, X_train_w2v, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

# print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [44]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 13.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.1   0.084 0.08  0.078 0.078 0.078 0.078 0.078 0.078 0.078 0.078
0.079
 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079
 0.08 ]
```

In [45]:

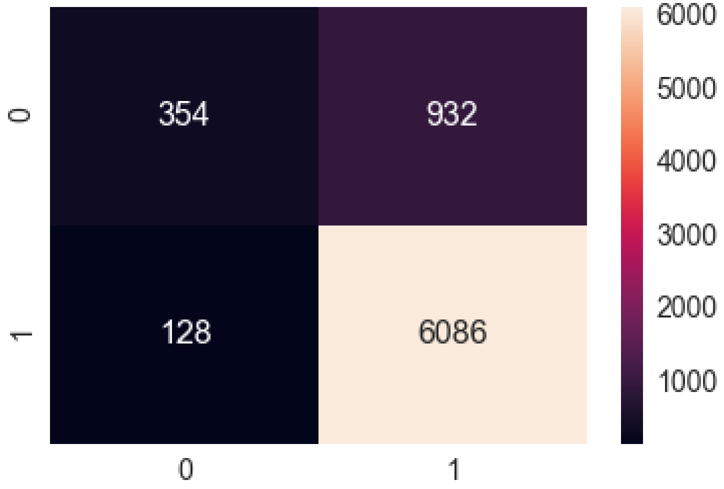
```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_w2v, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_w2v)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 13 is 0.830287
```

In [46]:
```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='brute')
knn.fit(X_train_w2v,y_train)
y_pred = knn.predict(X_test_w2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 85.867%
Precision on test set: 0.867
Recall on test set: 0.979
F1-Score on test set: 0.920
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

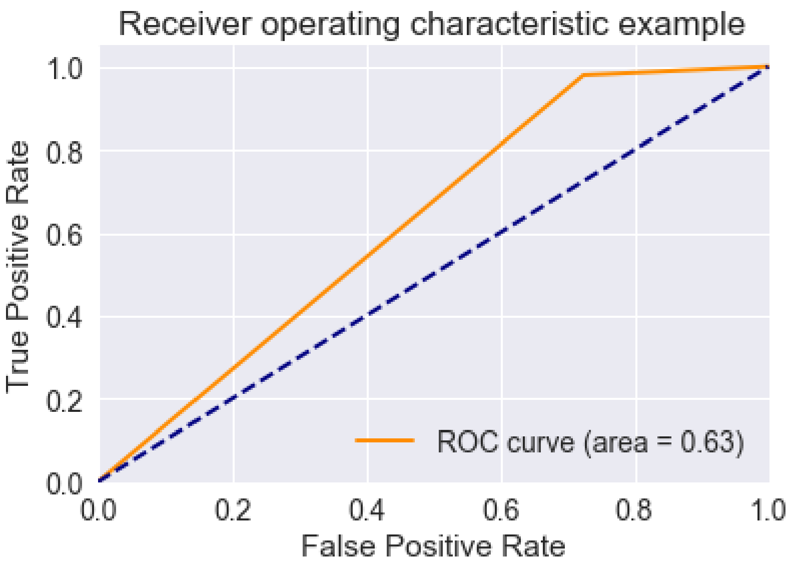
Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc3a6d32160>
```


0
1
354
932
128
6086
0
1
6000
5000
4000
3000
2000
1000

In [47]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.6273367567640622
```



TFIDF-W2V with Brute force

In [48]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(cleaned_data_sample['CleanedText'].values)
# 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 [49]:

```
# 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 list_of_sent: # 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
```

In [50]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidfw2v_vecs_norm = preprocessing.normalize(tfidf_sent_vectors)

#Not shuffling the data as we want it on time basis
X_train_tfw2v, X_test_tfw2v, y_train, y_test = train_test_split(tfidfw2v_vecs_norm,cleaned_data_sample['Score'].values,tes
```

In [51]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

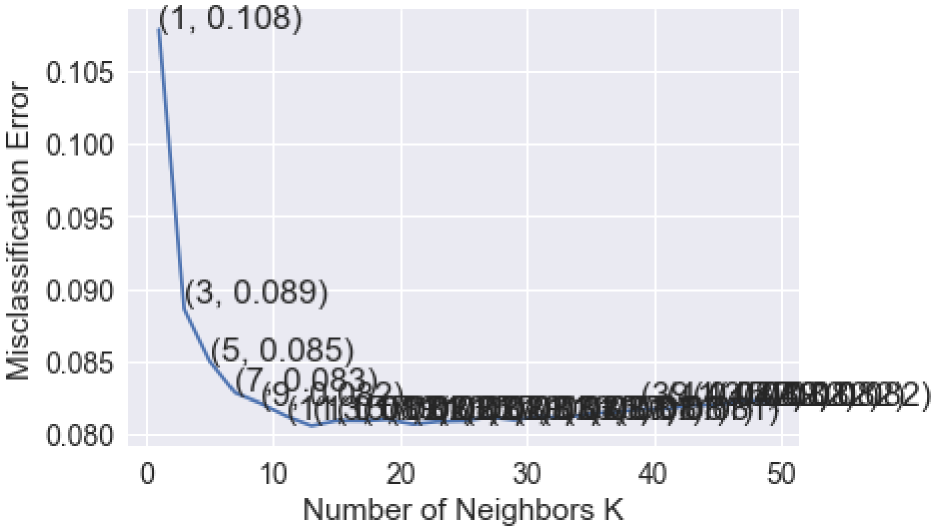
In [52]:

```
#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')

    score = cross_val_score(knn, X_train_tfw2v, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

#print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [53]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 13.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.108 0.089 0.085 0.083 0.082 0.081 0.081 0.081 0.081 0.081 0.081
0.081
 0.081 0.081 0.081 0.081 0.081 0.081 0.081 0.082 0.082 0.082 0.082 0.082
 0.082]
```

In [54]:

```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_tfw2v, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_tfw2v)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 13 is 0.816405
```

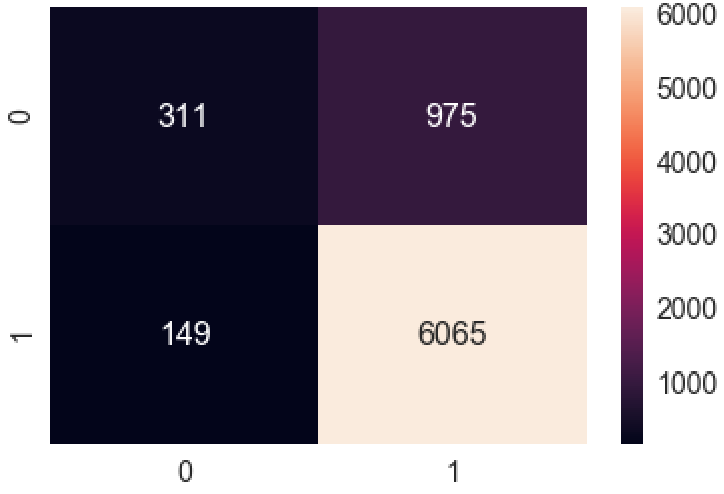
In [55]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='brute')
knn.fit(X_train_tfw2v,y_train)
y_pred = knn.predict(X_test_tfw2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 85.013%
Precision on test set: 0.862
Recall on test set: 0.976
F1-Score on test set: 0.915
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[55]:

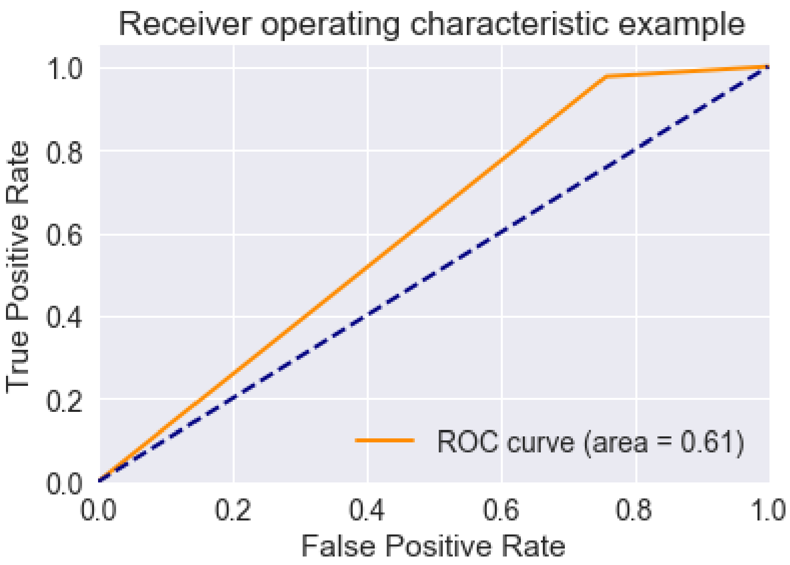
```
<matplotlib.axes._subplots.AxesSubplot at 0xc3a6d10c50>
```


311
975
149
6065
0
1
0
1
6000
5000
4000
3000
2000
1000

In [56]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.6089285169043362
```



BOW kdtree algo

In [21]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(cleaned_data_sample['CleanedText'].values,cleaned_data_sample['Score']

#BoW
count_vect = CountVectorizer(min_df=10,max_features=500) #in scikit-learn

X_train_bow = count_vect.fit_transform(X_train)
X_train_bow = preprocessing.normalize(X_train_bow)

X_test_bow = count_vect.transform(X_test)
X_test_bow = preprocessing.normalize(X_test_bow)

print("the type of count vectorizer for train data",type(X_train_bow))
print("the type of count vectorizer for test data",type(X_test_bow))

print("the shape of out train data BOW vectorizer ",X_train_bow.get_shape())
print("the number of unique words for train data", X_train_bow.get_shape()[1])

print("the shape of out test data BOW vectorizer ",X_test_bow.get_shape())
print("the number of unique words for test data", X_test_bow.get_shape()[1])
```

```
the type of count vectorizer for train data <class 'scipy.sparse.csr.csr_matrix'>
the type of count vectorizer for test data <class 'scipy.sparse.csr.csr_matrix'>
the shape of out train data BOW vectorizer  (17500, 500)
the number of unique words for train data 500
the shape of out test data BOW vectorizer  (7500, 500)
the number of unique words for test data 500
```

In [24]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

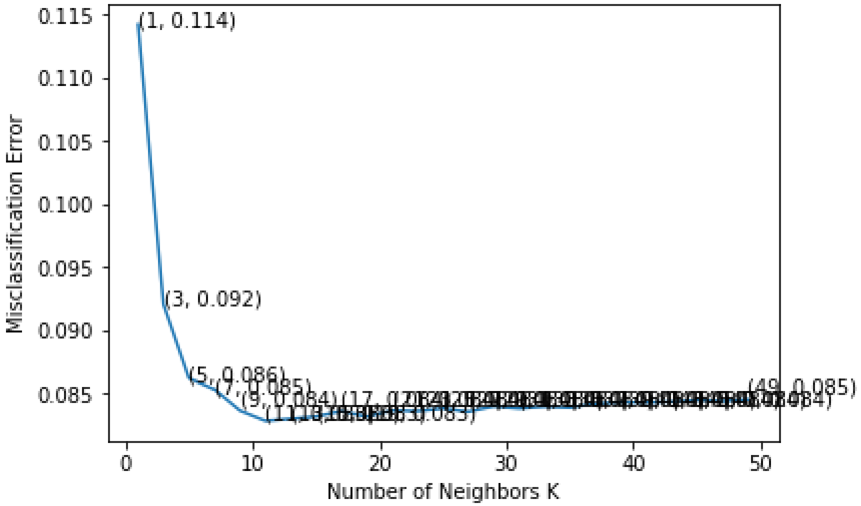
In [25]:

```
#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')

    score = cross_val_score(knn, X_train_bow, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

# print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [26]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 11.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.114 0.092 0.086 0.085 0.084 0.083 0.083 0.083 0.084 0.083 0.084
0.084
 0.084 0.084 0.084 0.084 0.084 0.084 0.084 0.084 0.084 0.084 0.084 0.084
 0.085]
```

In [27]:
```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_bow, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_bow)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 11 is 0.778186
```

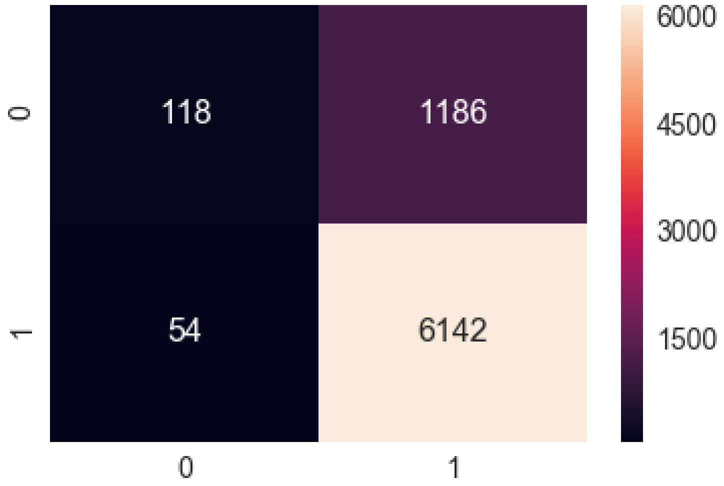
In [28]:

```

#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
knn.fit(X_train_bow,y_train)
y_pred = knn.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 83.467%
Precision on test set: 0.838
Recall on test set: 0.991
F1-Score on test set: 0.908
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

```

Out[28]:

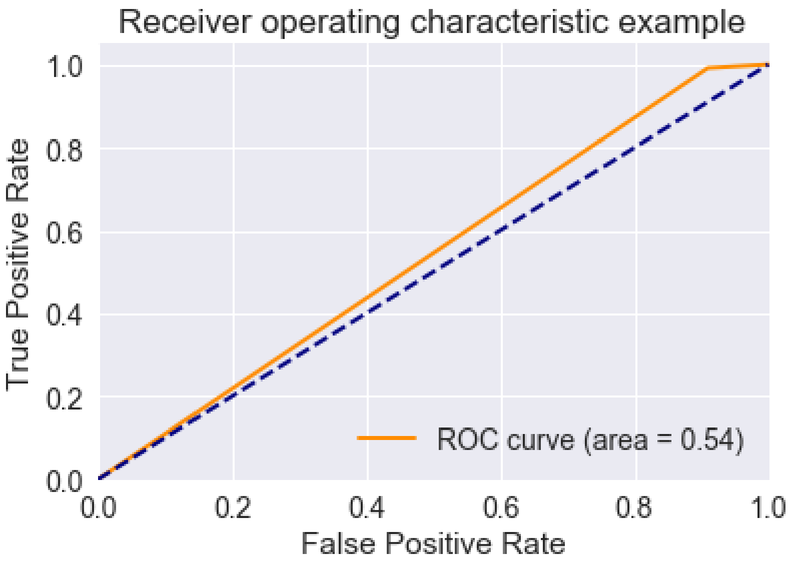
```
<matplotlib.axes._subplots.AxesSubplot at 0xad6c3e77f0>
```


118
1186
54
6142
0
1
0
1
6000
4500
3000
1500

In [29]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.5408877486761695
```



TF-IDF with kd tree

In [30]:

```
#Taking Sample Data
n_samples = 25000
cleaned_data_sample = cleaned_data.sample(n_samples)

###Sorting as we want according to time series
cleaned_data_sample.sort_values('Time',inplace=True)
```

In [31]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Breaking into Train and test
X_train, X_test, y_train, y_test = train_test_split(cleaned_data_sample['CleanedText'].values,cleaned_data_sample['Score']

tfidf = TfidfVectorizer(ngram_range=(1,2),min_df=10, max_features=500) #Using bi-grams
X_train_tfidf1 = tfidf.fit_transform(X_train)
#Normalize Data
X_train_tfidf1 = preprocessing.normalize(X_train_tfidf1)
print("Train Data Size: ",X_train_tfidf1.shape)
X_test_tfidf1 = tfidf.transform(X_test)
#Normalize Data
X_test_tfidf1 = preprocessing.normalize(X_test_tfidf1)
print("Test Data Size: ",X_test_tfidf1.shape)
```

```
Train Data Size:  (17500, 500)
Test Data Size:  (7500, 500)
```

In [32]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

In [33]:

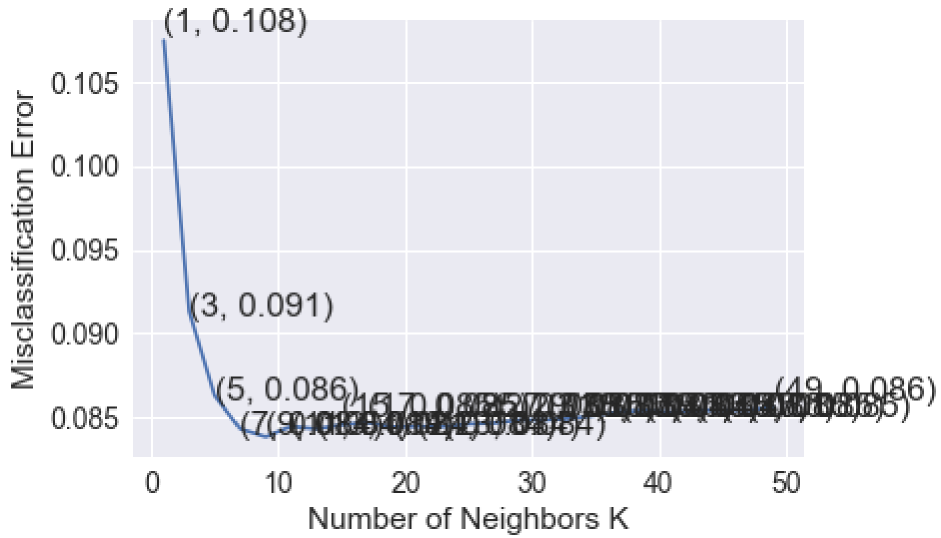
```

#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

np.array(X_train_tfidf1)
np.array(y_train)
np.array(X_train_tfidf1)
np.array(y_train)
np.array(X_test_tfidf1)

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')

    score = cross_val_score(knn, X_train_tfidf1, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

print('len(mis_class_error): ', len(mis_class_error))
```

```
len(cv_scores):  25
len(mis_class_error):  25
```

In [34]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 9.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.108 0.091 0.086 0.084 0.084 0.084 0.084 0.085 0.085 0.084 0.084
0.084
 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085 0.085
 0.086]
```

In [35]:

```
# =============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_tfidf1, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_tfidf1)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 9 is 0.778663
```

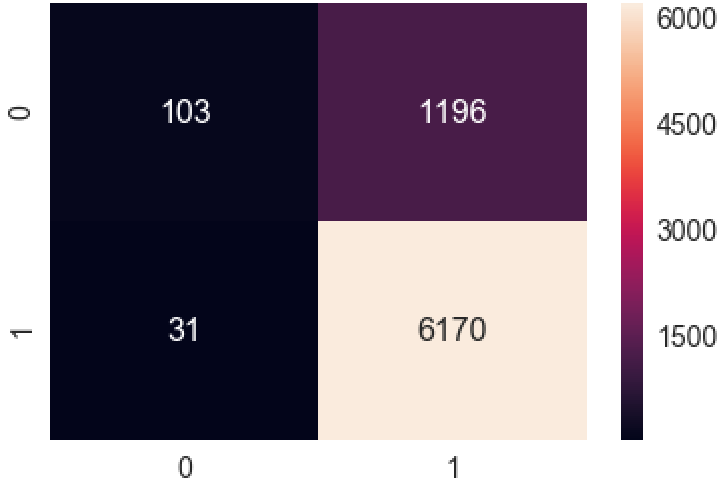
In [58]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
knn.fit(X_train_tfidf1,y_train)
y_pred = knn.predict(X_test_tfidf1)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 83.640%
Precision on test set: 0.838
Recall on test set: 0.995
F1-Score on test set: 0.910
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad69ba1a58>
```

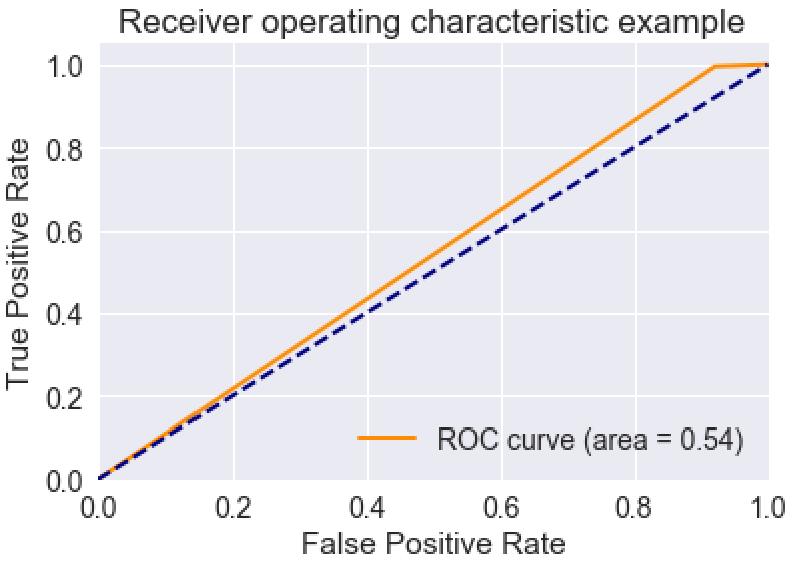


103
1196
31
6170
0
1
0
1
6000
4500
3000
1500

In [59]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.5371462846080477
```



avg w2v with kd tree

In [37]:

```
i=0
list_of_sent=[]
for sent in cleaned_data_sample['CleanedText'].values:
    list_of_sent.append(sent.split())
```

In [38]:

```
print(cleaned_data_sample['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])
```

```
realli good idea final product outstand use decal car window everybodi ask bought decal made two thumb
*****************************************************************
['realli', 'good', 'idea', 'final', 'product', 'outstand', 'use', 'decal', 'car', 'window', 'everybodi', 'ask', 'bough
t', 'decal', 'made', 'two', 'thumb']
```

In [39]:

```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [40]:

```
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  6501
sample words  ['realli', 'good', 'idea', 'final', 'product', 'outstand', 'use', 'car', 'window', 'everybodi', 'ask',
'bought', 'made', 'two', 'thumb', 'receiv', 'shipment', 'could', 'hard', 'wait', 'tri', 'love', 'call', 'instead', 'st
icker', 'remov', 'easili', 'daughter', 'design', 'sign', 'print', 'revers', 'beauti', 'shop', 'program', 'go', 'lot',
'fun', 'everywher', 'surfac', 'like', 'screen', 'comput', 'monitor', 'noth', 'bother', 'link', 'top', 'page', 'buy']
```

In [41]:
```
# Average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    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]))
sent_vectors = np.array(sent_vectors)
```

```
25000
50
```

In [42]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

avg_vec_norm = preprocessing.normalize(sent_vectors)

#Not shuffling the data as we want it on time basis
X_train_w2v1, X_test_w2v1, y_train, y_test = train_test_split(avg_vec_norm,cleaned_data_sample['Score'].values,test_size=0
```

In [43]:
```
avg_vec_norm.shape
```

Out[43]:
```
(25000, 50)
```

In [44]:

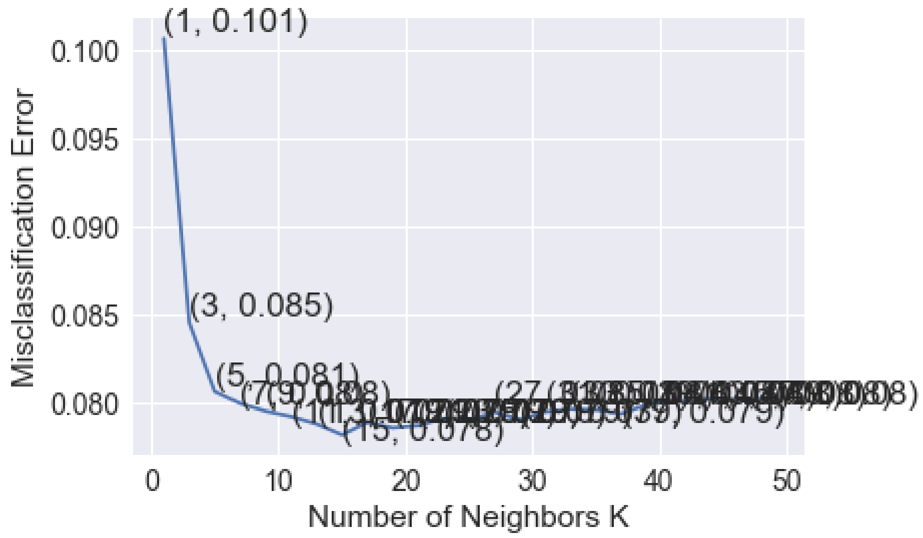
```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

In [45]:

```
#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')

    score = cross_val_score(knn, X_train_w2v1, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

# print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [46]:
```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 15.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.101 0.085 0.081 0.08  0.08  0.079 0.079 0.078 0.079 0.079 0.079
0.079
 0.079 0.08  0.079 0.08  0.08  0.08  0.079 0.08  0.08  0.08  0.08  0.08
 0.08 ]
```

In [47]:

```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_w2v1, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_w2v1)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 15 is 0.831314
```

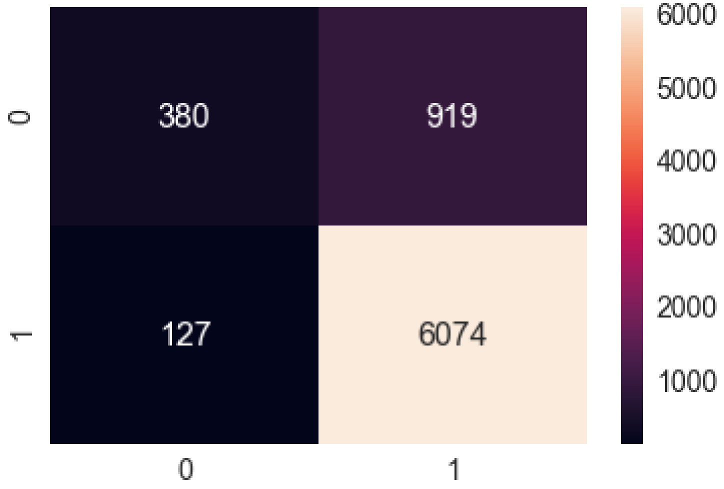
In [61]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
knn.fit(X_train_w2v1,y_train)
y_pred = knn.predict(X_test_w2v1)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 86.053%
Precision on test set: 0.869
Recall on test set: 0.980
F1-Score on test set: 0.921
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

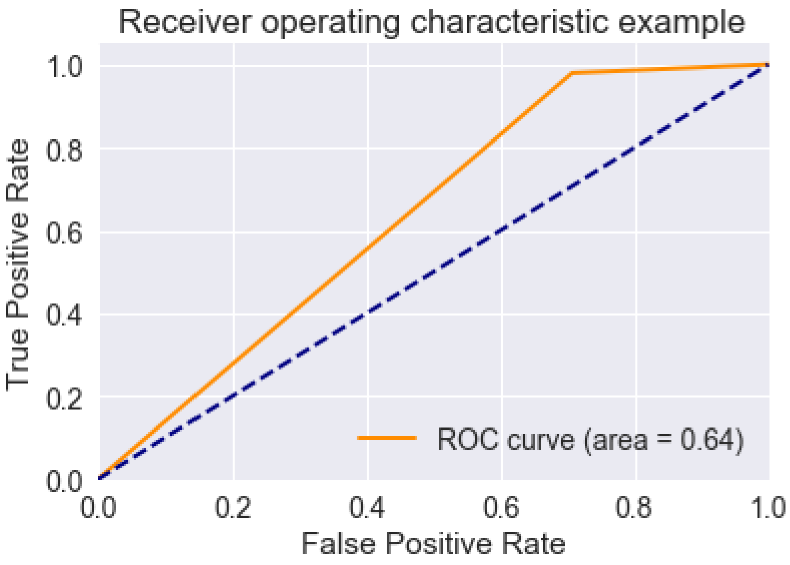
Out[61]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xad6c354780>
```


380
919
127
6074
0
1
0
1
6000
5000
4000
3000
2000
1000

In [62]:
```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.6360260749123009
```



tfidf W2V kd tree

In [49]:

```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(cleaned_data_sample['CleanedText'].values)
# 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 [50]:

```
# 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
```

In [51]:

```
tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in list_of_sent: # 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
```

In [52]:

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

tfidfw2v_vecs_norm = preprocessing.normalize(tfidf_sent_vectors)

#Not shuffling the data as we want it on time basis
X_train_w2v2, X_test_w2v2, y_train, y_test = train_test_split(tfidfw2v_vecs_norm,cleaned_data_sample['Score'].values,test_
```

In [53]:

```
# split the train data set into cross validation train and cross validation test
#X_cv, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)

from sklearn.model_selection import TimeSeriesSplit
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

tss = TimeSeriesSplit(n_splits=10)

# creating odd list of K values for KNN
neighbors = [y for y in range(1,50,2)]

# empty lists to hold cross-validation scores and misclassification errors
cv_scores = []
mis_class_error = []
```

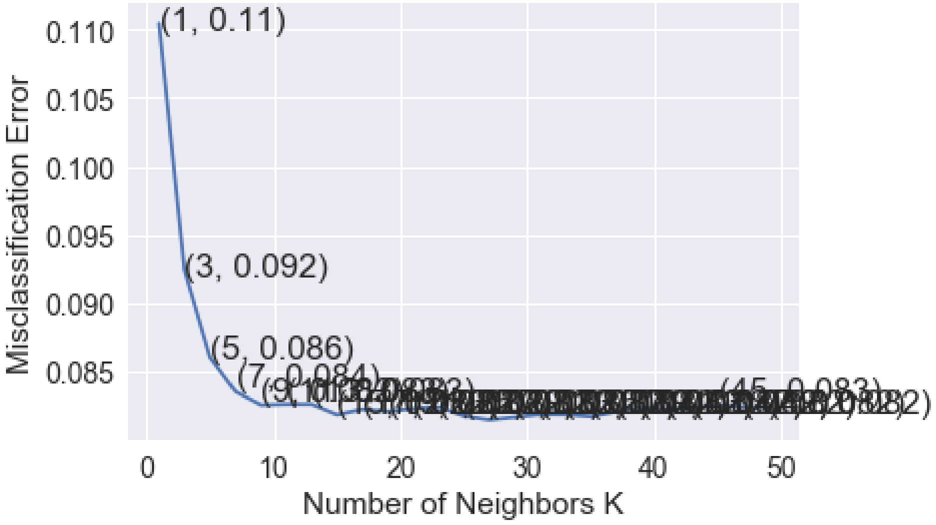
In [54]:

```
#looping through all the K values in 'neighbours' list and calculating the following:
# 1. Cross-validation scores- for each K value we get 10 values of cross-validation.
#    Calculate the mean of the cross-validation scores and append it in cv_scores list
# 2. After the loop calculate the misclassification error for all the cv_scores values
from sklearn.model_selection import cross_val_score

for i in (neighbors):

    knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')

    score = cross_val_score(knn, X_train_w2v2, y_train, cv=tss, scoring='f1')
#     print(score)
    cv_scores.append(score.mean())

# print('len(cv_scores): ', len(cv_scores))

mis_class_error = [1-x for x in cv_scores]

# print('len(mis_class_error): ', len(mis_class_error))
```

In [55]:

```
# determining best K value using minimum misclassification error value
# and picking the corresponding K value from neighbour list

optimal_k = neighbors[mis_class_error.index(min(mis_class_error))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)

print('len(neighbors): ', len(neighbors))

print('len(mis_class_error): ', len(mis_class_error))

# plotting misclassification error vs k
plt.plot(neighbors, mis_class_error)

for xy in zip(neighbors, np.round(mis_class_error,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(mis_class_error,3))
```

```
The optimal number of neighbors is 27.
len(neighbors):  25
len(mis_class_error):  25
```



```
the misclassification error for each k value is :  [0.11  0.092 0.086 0.084 0.083 0.083 0.083 0.082 0.082 0.082 0.082
0.082
 0.082 0.082 0.082 0.082 0.082 0.082 0.082 0.082 0.082 0.082 0.083 0.082
 0.082]
```

In [56]:
```
# ============================== KNN with k = optimal_k ===============================================
# instantiate learning model K = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model with optimal K for training data
knn_optimal.fit(X_train_w2v2, y_train)


# predict the response for the unseen bow_test data
pred = knn_optimal.predict(X_test_w2v2)

# evaluate f1 score for the prediction
from sklearn.metrics import f1_score
acc = f1_score(y_test, pred,average='weighted')
print('\nThe accuracy of the knn classifier for k = %d is %f' % (optimal_k, acc))
```

```
The accuracy of the knn classifier for k = 27 is 0.811901
```

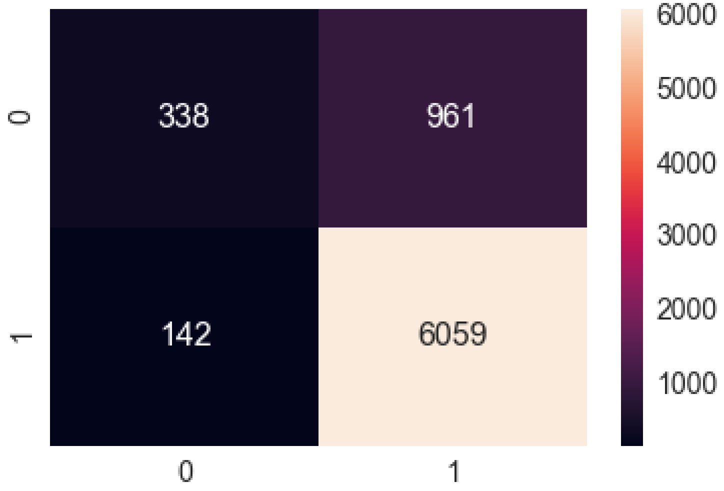
In [63]:

```
#Testing Accuracy on Test data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

knn = KNeighborsClassifier(n_neighbors=11,algorithm='kd_tree')
knn.fit(X_train_w2v2,y_train)
y_pred = knn.predict(X_test_w2v2)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 85.293%
Precision on test set: 0.863
Recall on test set: 0.977
F1-Score on test set: 0.917
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[63]:

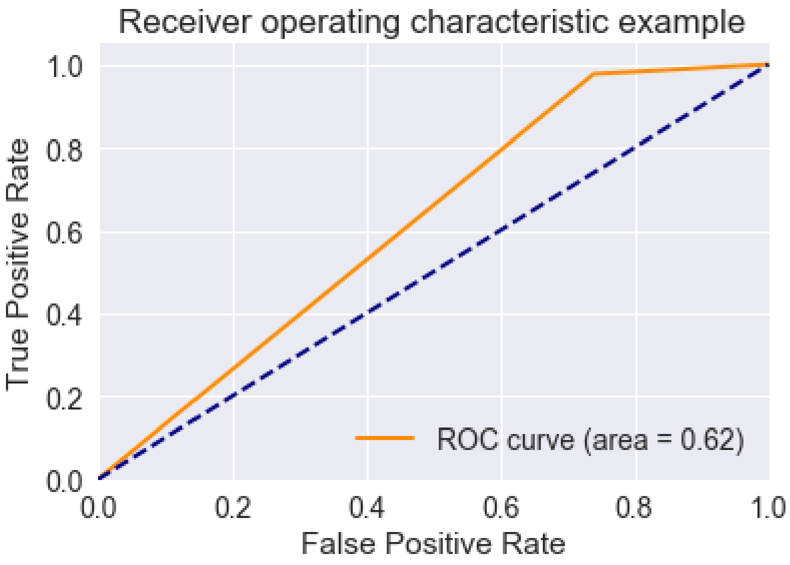
```
<matplotlib.axes._subplots.AxesSubplot at 0xad0372d780>
```


338
961
142
6059
0
1
0
1
6000
5000
4000
3000
2000
1000

In [64]:

```
import numpy as np
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
fpr, tpr, thresholds = metrics.roc_curve(y_test,y_pred)
metrics.auc(fpr,tpr)
roc_auc =auc(fpr,tpr)
print('AUC =',auc(fpr,tpr))
plt.figure
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
AUC = 0.6186503108155468
```



[6] Conclusions

```
1. The KNN algorithm has been applied here only on 25k data, therefore not exact accurate results have been obtained.
2. The KNN algorithm takes more time, therefore more time latency.
3. There is not much difference between the accuracy on the training data and the test data. Therefore, the model is
working almost accurate for all the sets.
4. The Kd_tree algorithm takes much time compared to the brute force.
5. Both kd_tree as well as Brute force gives almost similar results.
6. The accuracy on all the sets is almost the same ~ 80%
7. Overall, KNN doesnt gives accurate results ,precision is almost ~ 85%, recall is ~ .95 . The F1- score (the weighted
average of recall and precision) is almost ~ 90% which is high. Even though the F1-score is high, we cant rely on this
result as the data taken here is imbalanced and less compared to real world. Therefore KNN is not recommended.
```