Amazon Fine Food Reviews Preprocessing

This IPython notebook consists code for preprocessing of text, conversion of text into vectors and saving that information for further use.

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

Public Information -

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

1. Number of reviews: 568,454
2. Number of users: 256,059
3. Number of products: 74,258
4. Timespan: Oct 1999 - Oct 2012
5. 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

Current Objective -

Go through the reviews and perform preprocessing, convert them into vectors and save them for future use.

[1] Reading Data

[1.1] Loading data and libraries

In [1]:

```
#importing necessary libraries
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import missingno as msno
import seaborn as sns

from nltk.stem.wordnet import WordNetLemmatizer
import re
from nltk.corpus import stopwords
from nltk import pos_tag, word_tokenize

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import nltk
import pickle

from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from gensim.models import Word2Vec
from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor
from concurrent import futures

from numba import jit

import os
from tqdm import tqdm
```

In [2]:

```
# setting path
par_path = os.path.normpath(os.getcwd() + os.sep + os.pardir)
dir_path = os.path.join(par_path, '0.datasets', 'pkl_dumps')
dir_path
```

Out[2]:

```
'D:\\appliedAI\\0.assignments\\0.datasets\\pkl_dumps'
```

In [10]:

```
#connecting to sqlite db
con = sqlite3.connect(os.path.join(dir_path, 'database.sqlite'))

#filtering only positive and negative reviews
data = pd.read_sql_query("SELECT * FROM Reviews WHERE Score != 3", con)

print("Shape of data:", data.shape)

#scores < 3 are considered to be negative reviews and > 3 are considered to be positive rev
data.head()
```

```
Shape of data: (525814, 10)
```

Out[10]:

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

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Missing values

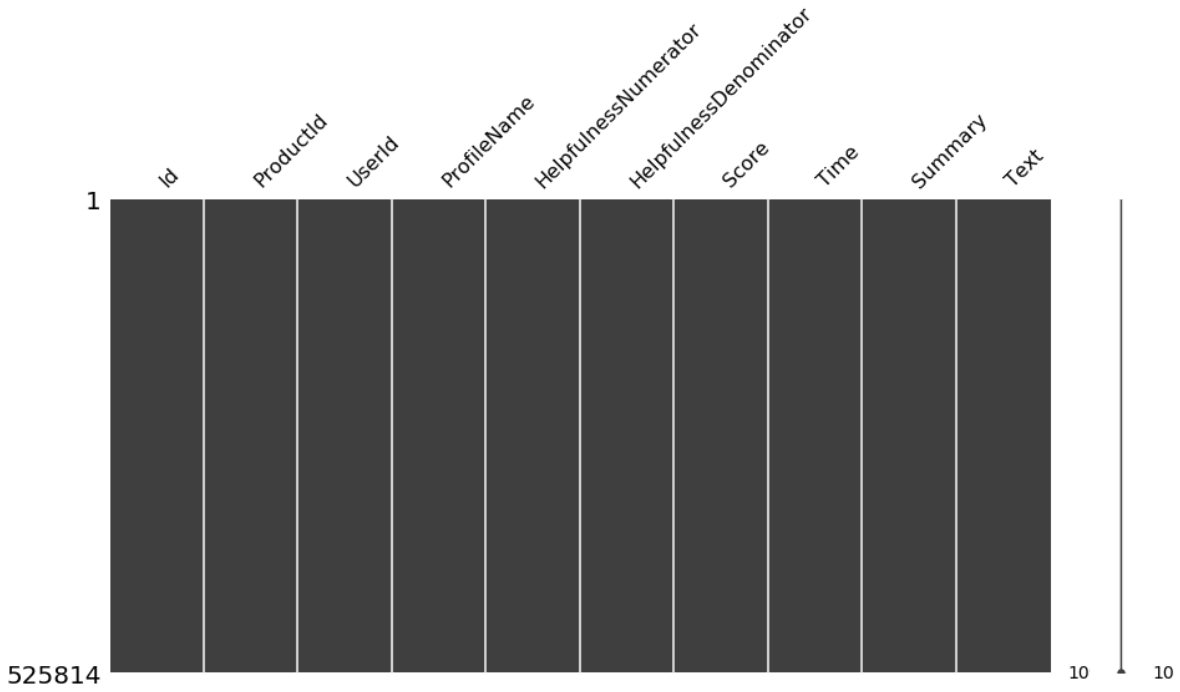
In [11]:

```
#let's just check, just in case if any
print("Missing values? Ans -", data.isnull().values.any())

#visualizing it
msno.matrix(data, figsize=(15,7))
```

```
Missing values? Ans - False
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f6bb6518>
```



[2.2] Data cleaning: Multiple reviews for the same product by same person

In [12]:

```
df = data.copy()
df['ProdUser'] = df['ProductId'] + df['UserId']
df[df['ProdUser'].duplicated(keep=False)].sort_values(
    'ProdUser', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last'
)
```

Out[12]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominat
157863	171174	7310172001	AE9ZBY7WW3LIQ	W. K. Ota	0	
157871	171183	7310172001	AE9ZBY7WW3LIQ	W. K. Ota	5	
157912	171228	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	5	
157841	171152	7310172001	AJD41FBJD9010	N. Ferguson "Two, Daisy, Hannah, and Kitten"	0	

Obeservations

1. There are some instances where a user has written more than one review for the same product.
2. We can remove the one which has less Helpfulness but lets keep all and treat it as review from a different user.
3. Will definitely have to remove same reviews because it is just redundant data.

In [13]:

```
#Sorting data according to ProductId in ascending order
data = data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='quicksort
```

[2.3] Data cleaning: Deduplication - 1

In [14]:

```
#Deduplication of entries
data=data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
data.shape
```

Out[14]:

```
(364173, 10)
```

In [15]:

```
data.head(2)
```

Out[15]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	

[2.4] Data cleaning: Deduplication - 2

Same reviews on multiple products with different timestamps

In [16]:

```
data[data['Text'].duplicated(keep=False)].sort_values(
    'Text', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last'
)
```

Out[16]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenor
67574	73444	B0046IISFG	A3OXHLG6DIBRW8	C. F. Hill "CFH"	1	
287090	311004	B001EO6FPU	A3OXHLG6DIBRW8	C. F. Hill "CFH"	9	
302818	327982	B0000CEQ6H	A281NPSIMI1C2R	Rebecca of Amazon "The	3	

In [17]:

```
#removing duplicate reviews
data=data.drop_duplicates(subset={"Text"}, keep='first', inplace=False)
data.shape
```

Out[17]:

```
(363836, 10)
```

Observations

1. There are reviews which are same on similar products (mostly different flavors).
2. These reviews were posted with different timestamps by the same person (weird).
3. Since we are interested in a review being positive or negative, having redundant reviews makes no sense, so removing them.

[2.5] Data cleaning: Removing practically impossible data

In [18]:

```
#also removing those reviews where HelpfulnessNumerator is greater than HelpfulnessDenomina
data=data[data['HelpfulnessNumerator']<=data['HelpfulnessDenominator']]
data.shape
```

Out[18]:

```
(363834, 10)
```

In [19]:

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

In [20]:

```
actualScore = data['Score']
positiveNegative = actualScore.map(partition)
data['Score'] = positiveNegative
print("Negatives shape:", data[data['Score']=='negative'].shape)
print("Positives shape:", data[data['Score']=='positive'].shape)
```

```
Negatives shape: (57070, 10)
Positives shape: (306764, 10)
```

[3] Text Preprocessing

We will be doing the following in order.

1. Text cleaning - includes removal of special characters which are not required.
2. Check if the word is actually an English word.

3. 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)
4. Convert the word to lower case.
5. Remove stop words but let's keep words like 'not' which makes the sentence negative.
6. POS Tagging and WordNet Lemmatizing the word.

In [21]:

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

In [22]:

```
stop = list(set(stopwords.words('english'))) #set of stopwords
print(stop)

#removing words like 'not' that gives negative meaning to a sentence from stopwords
important_stopwords = ['hadn', 'weren', 'shouldn', "needn't", 'needn', 'doesn', "shan't", "
                       "wasn", "couldn't", 'mustn', "hadn't", "doesn't", "don't", "not",
                      'wouldn', "weren't", "didn", "mustn't", "wasn't", "didn't"]
pre_final_stops = [x for x in stop if x not in important_stopwords]

#removing punctuation from stop words
final_stops = list(set([cleanpunc(x) for x in pre_final_stops]))
print("Final stopwords:", final_stops)
```

```
['our', 'has', 'an', 'as', 'then', 'here', "doesn't", 'aren', 'she', 'whil
e', 'out', "won't", "you'd", 'haven', 'him', 'was', 'against', 'more', 'whe
n', 'above', 'through', 'didn', 'for', 'during', 'on', "wasn't", 'hadn', 'm
y', 'are', 'himself', "isn't", 'not', 'herself', 'both', 'm', 'ma', "should
n't", 'your', 'y', 'isn', 'some', 'mightn', 'at', 'why', 'o', 'can', 'betwee
n', 've', "you're", 'this', 'of', 'just', "aren't", "haven't", 'theirs', 'm
e', 'up', 'down', 'it', 'have', 'had', 'do', 'we', 'other', "mustn't", 'ver
y', 'couldn', 'won', 'they', 'with', 'he', 'having', 'only', 'don', 'his',
"you'll", "she's", 'by', 'but', 'myself', 'will', 'before', 'until', 'such',
'its', 'now', 'over', 'll', 'd', 'from', 'where', 'too', 'a', 'any', "had
n't", "it's", 'after', "mightn't", 'how', 'because', 'in', 'be', 'should',
'doing', 'few', 'to', 'needn', 'ain', 'shouldn', 'shan', 'the', 'her', "were
n't", "shan't", 'nor', 'that', "hasn't", 'were', 'does', 'mustn', 'hasn', 'y
ourself', 'no', 'under', 'and', 'all', 'did', 'been', "couldn't", 'which',
's', 'is', "should've", 'doesn', 'each', 'you', 'same', "that'll", 'who', "y
ou've", 'those', 'than', 're', 'own', 'being', 'off', "didn't", "don't", 'wo
uldn', 'ours', "wouldn't", 'so', 'or', 'these', 'yours', 'yourselves', 'if',
'themselves', 'i', 'am', 'what', 'into', 'again', 'further', 'weren', 'itsel
f', 'below', 't', 'ourselves', 'hers', 'them', 'whom', 'about', 'most', 'the
re', 'once', "needn't", 'wasn', 'their']
Final stopwords: ['has', 'our', 'an', 'as', 'then', 'here', 'aren', 'she',
'while', 'out', 'haven', 'him', 'was', 'against', 'more', 'when', 'above',
'through', 'for', 'during', 'on', 'my', 'are', 'himself', 'herself', 'both',
'm', 'youve', 'ma', 'your', 'wont', 'havent', 'y', 'shes', 'isn', 'some', 'm
ightn', 'at', 'why', 'o', 'can', 'between', 've', 'this', 'of', 'just', 'wou
ldnt', 'mightnt', 'theirs', 'me', 'up', 'down', 'it', 'have', 'had', 'do',
'we', 'other', 'very', 'arent', 'couldn', 'won', 'they', 'with', 'he', 'havi
ng', 'only', 'don', 'his', 'by', 'but', 'myself', 'will', 'before', 'until',
'such', 'its', 'now', 'hasnt', 'over', 'youd', 'll', 'd', 'from', 'where',
'isnt', 'too', 'a', 'any', 'after', 'how', 'because', 'in', 'be', 'should',
'doing', 'few', 'to', 'ain', 'shan', 'the', 'her', 'nor', 'that', 'were', 'd
oes', 'hasn', 'yourself', 'no', 'under', 'and', 'all', 'did', 'been', 'whic
h', 's', 'is', 'each', 'you', 'same', 'who', 'those', 'than', 're', 'own',
'being', 'off', 'ours', 'so', 'or', 'these', 'yours', 'yourselves', 'if', 's
houldve', 'themselves', 'i', 'am', 'what', 'thatll', 'into', 'again', 'furth
er', 'itself', 'below', 'youll', 'youre', 't', 'ourselves', 'hers', 'them',
'whom', 'about', 'most', 'there', 'once', 'their']
```

In [23]:

```
wnl = WordNetLemmatizer()
```

In [25]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
scores = data['Score'].values
for sent in tqdm(data['Text'].values):
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    tokens = pos_tag(word_tokenize(sent))
    for w in tokens:
        for cleaned_words in cleanpunc(w[0]).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    #s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    # lemmatization works better with POS tagging
                    tag = w[1][0].lower()
                    tag = tag if tag in ['a', 'n', 'v'] else None
                    if not tag:
                        s = cleaned_words.lower().encode('utf8')
                    else:
                        s = wnl.lemmatize(cleaned_words.lower(), tag).lower().encode("utf8"
                    filtered_sentence.append(s)
                    if scores[i] == "positive":
                        all_positive_words.append(s) #list of all words used to describe po
                    if scores[i] == "negative":
                        all_negative_words.append(s) #list of all words used to describe ne
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("**********************************************************************")

    final_string.append(str1)
    i+=1
print("Done!")
```

```
100%|████████████████████████████████████████████████████████████████
██| 363834/363834 [53:56<00:00, 112.42it/s]
```

Done!

In [26]:

```
data['CleanedText']=final_string #adding a column of CleanedText which displays the data af
data['CleanedText']=data['CleanedText'].str.decode("utf-8")
```

In [27]:

```
# store final table into an SQLite table for future.
conn = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
c=conn.cursor()
conn.text_factory = str
data.to_sql('Reviews', conn,  schema=None, if_exists='replace', index=True, index_label=Nor
```

In [3]:

```
##############################################################################
##############################################################################
con = sqlite3.connect(os.path.join(dir_path, 'final.sqlite'))
data = pd.read_sql_query(""" SELECT * FROM Reviews """, con)
del data['index']
data.shape
```

Out[3]:

```
(363834, 11)
```

In [29]:

```
data.head()
```

Out[29]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [30]:

```
data['Time'] = data['Time'].astype('int')
data.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort', na_positi
```

Out[30]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDen
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
30	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
424	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	

In [8]:

```
train_df = data.head(60000).copy()
cv_df = data[60000:80000].copy()
test_df = data[80000:100000].copy()
```

[4] Featurization

[4.1] Bag of words - unigrams and bigrams

In [32]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
train_final_counts = count_vect.fit_transform(train_df['CleanedText'].values)
cv_final_counts = count_vect.transform(cv_df['CleanedText'].values)
test_final_counts = count_vect.transform(test_df['CleanedText'].values)
print("the type of count vectorizer ",type(train_final_counts))
print("the shape of out text BOW vectorizer ",train_final_counts.get_shape())
print("the number of unique words ", train_final_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (60000, 37918)
the number of unique words  37918
```

In [33]:

```
freq_dist_positive=nltk.FreqDist(all_positive_words)
freq_dist_negative=nltk.FreqDist(all_negative_words)
print("Most Common Positive Words : ",freq_dist_positive.most_common(20))
print("Most Common Negative Words : ",freq_dist_negative.most_common(20))
```

```
Most Common Positive Words :  [(b'like', 137224), (b'taste', 122048), (b'goo
d', 111392), (b'love', 104941), (b'great', 103358), (b'use', 101467), (b'mak
e', 100402), (b'flavor', 99414), (b'one', 96800), (b'get', 93102), (b'produc
t', 90812), (b'try', 86223), (b'tea', 82861), (b'coffee', 78958), (b'find',
78423), (b'buy', 75963), (b'food', 64946), (b'would', 59997), (b'eat', 5744
1), (b'time', 54081)]
Most Common Negative Words :  [(b'taste', 33523), (b'like', 31734), (b'produ
ct', 28122), (b'buy', 20800), (b'one', 20593), (b'would', 20028), (b'get', 2
0000), (b'flavor', 18124), (b'try', 17575), (b'make', 16240), (b'use', 1491
5), (b'good', 14894), (b'coffee', 14764), (b'order', 12792), (b'food', 1275
6), (b'think', 11931), (b'tea', 11634), (b'eat', 11014), (b'even', 10947),
(b'box', 10812)]
```

In [35]:

```
#saving BoW unigrams
with open(os.path.join(dir_path, "bow_uni_vec_train.pkl"), 'wb') as bow:
    pickle.dump(train_final_counts, bow)
with open(os.path.join(dir_path, "train_lab.pkl"), 'wb') as bow:
    pickle.dump(train_df['Score'].values, bow)

with open(os.path.join(dir_path, "bow_uni_vec_cv.pkl"), 'wb') as bow:
    pickle.dump(cv_final_counts, bow)
with open(os.path.join(dir_path, "cv_lab.pkl"), 'wb') as bow:
    pickle.dump(cv_df['Score'].values, bow)

with open(os.path.join(dir_path, "bow_uni_vec_test.pkl"), 'wb') as bow:
    pickle.dump(test_final_counts, bow)
with open(os.path.join(dir_path, "test_lab.pkl"), 'wb') as bow:
    pickle.dump(test_df['Score'].values, bow)
```

In [36]:

```
#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) ) #in scikit-learn
train_bigram_counts = count_vect.fit_transform(train_df['CleanedText'].values)
cv_bigram_counts = count_vect.transform(cv_df['CleanedText'].values)
test_bigram_counts = count_vect.transform(test_df['CleanedText'].values)
print("the type of count vectorizer ",type(train_bigram_counts))
print("the shape of out text BOW vectorizer ",train_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", train_bigram_count
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (60000, 905447)
the number of unique words including both unigrams and bigrams  905447
```

In [37]:

```
#saving BoW bigrams
with open(os.path.join(dir_path, "bow_bi_vec_train.pkl"), 'wb') as bow:
    pickle.dump(train_bigram_counts, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_
#     pickle.dump(train_df['Score'].values, bow)

with open(os.path.join(dir_path, "bow_bi_vec_cv.pkl"), 'wb') as bow:
    pickle.dump(cv_bigram_counts, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_
#     pickle.dump(cv_df['Score'].values, bow)

with open(os.path.join(dir_path, "bow_bi_vec_test.pkl"), 'wb') as bow:
    pickle.dump(test_bigram_counts, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/bow_bi_vec_
#     pickle.dump(test_df['Score'].values, bow)
```

[4.2] TF-IDF

In [38]:

```
#tf-idf
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
train_tf_idf = tf_idf_vect.fit_transform(train_df['CleanedText'].values)
cv_tfidf = tf_idf_vect.transform(cv_df['CleanedText'].values)
test_tfidf = tf_idf_vect.transform(test_df['CleanedText'].values)
print("the type of count vectorizer ",type(train_tf_idf))
print("the shape of out text TFIDF vectorizer ",train_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", train_tf_idf.get_s
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (60000, 905447)
the number of unique words including both unigrams and bigrams  905447
```

In [39]:

```
features = tf_idf_vect.get_feature_names()
print("some sample features(unique words in the corpus)",features[100000:100010])
```

```
some sample features(unique words in the corpus) ['broccoli pepper', 'brocco
li plain', 'broccoli powder', 'broccoli precook', 'broccoli quietly', 'brocc
oli really', 'broccoli recommend', 'broccoli red', 'broccoli rice', 'broccol
i run']
```

In [40]:

```
def top_tfidf_feats(row, features, top_n=25):
    ''' Get top n tfidf values in row and return them with their corresponding feature name
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

top_tfidf = top_tfidf_feats(train_tf_idf[1,:].toarray()[0],features,25)
```

In [41]:

```
top_tfidf
```

Out[41]:

	feature	tfidf
0	paperback seem	0.181921
1	page open	0.181921
2	movie incorporate	0.181921
3	read sendak	0.181921
4	rosie movie	0.181921
5	version paperback	0.181921
6	keep page	0.181921
7	incorporate love	0.181921
8	cover version	0.181921
9	watch really	0.175399
10	really rosie	0.175399
11	two hand	0.175399
12	however miss	0.175399
13	sendak book	0.175399
14	kind flimsy	0.175399
15	book watch	0.175399
16	miss hard	0.175399
17	flimsy take	0.175399
18	grow read	0.175399
19	hard cover	0.175399
20	paperback	0.167181
21	rosie	0.164248
22	seem kind	0.164248
23	hand keep	0.157726
24	love son	0.157726

In [42]:

```
#saving tfidf
with open(os.path.join(dir_path, "tfidf_vec_train.pkl"), 'wb') as bow:
    pickle.dump(train_tf_idf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_t
#     pickle.dump(train_df['Score'].values, bow)

with open(os.path.join(dir_path, "tfidf_vec_cv.pkl"), 'wb') as bow:
    pickle.dump(cv_tfidf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_c
#     pickle.dump(cv_df['Score'].values, bow)

with open(os.path.join(dir_path, "tfidf_vec_test.pkl"), 'wb') as bow:
    pickle.dump(test_tfidf, bow)
# with open("/content/gdrive/My Drive/appliedAI/datasets/amzn_fine_food_reviews/tfidf_vec_t
#     pickle.dump(test_df['Score'].values, bow)
```

In [43]:

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

[4.3] Word2Vec

In [44]:

```
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=7)
```

In [45]:

```
#saving w2v model
w2v_model.save(os.path.join(dir_path, "amzn_w2v_vec.model"))
```

In [4]:

```
#loading model
w2v_model = Word2Vec.load(os.path.join(dir_path, "amzn_w2v_vec.model"))
```

In [5]:

```
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  12106
sample words  ['little', 'book', 'make', 'son', 'laugh', 'loud', 'recite',
'car', 'drive', 'along', 'always', 'sing', 'refrain', 'learn', 'whale', 'ind
ia', 'droop', 'rose', 'love', 'new', 'word', 'classic', 'willing', 'bet', 's
till', 'able', 'memory', 'college', 'grow', 'read', 'sendak', 'watch', 'real
ly', 'rosie', 'movie', 'incorporate', 'however', 'miss', 'hard', 'cover', 'v
ersion', 'paperback', 'seem', 'kind', 'flimsy', 'take', 'two', 'hand', 'kee
p', 'page']
```

[4.3.1] Average Word2Vec

In [48]:

```
# average Word2Vec
# compute average word2vec for each review.
def avg_w2vec(list_of_sent):
    sent_vectors = [] # the avg-w2v for each sentence/review is stored in this list
    for sent in tqdm(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]))
    return sent_vectors
```

In [49]:

```
avg_w2v_train = avg_w2vec([sent.split() for sent in train_df['CleanedText'].values])
avg_w2v_cv = avg_w2vec([sent.split() for sent in cv_df['CleanedText'].values])
avg_w2v_test = avg_w2vec([sent.split() for sent in test_df['CleanedText'].values])
```

```
100%|████████████████████████████████████████████████████████████████
████| 60000/60000 [02:09<00:00, 462.21it/s]

60000
50

100%|████████████████████████████████████████████████████████████████
████| 20000/20000 [00:41<00:00, 485.60it/s]

20000
50

100%|████████████████████████████████████████████████████████████████
████| 20000/20000 [00:40<00:00, 498.35it/s]

20000
50
```

In [50]:

```
#saving word2vec
with open(os.path.join(dir_path, "avg_w2v_train.pkl"), 'wb') as w2v_pickle:
    pickle.dump(avg_w2v_train, w2v_pickle)
with open(os.path.join(dir_path, "avg_w2v_cv.pkl"), 'wb') as w2v_pickle:
    pickle.dump(avg_w2v_cv, w2v_pickle)
with open(os.path.join(dir_path, "avg_w2v_test.pkl"), 'wb') as w2v_pickle:
    pickle.dump(avg_w2v_test, w2v_pickle)
```

[4.3.2] TFIDF-Word2Vec

In [51]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(train_df['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 [52]:

```
# TF-IDF weighted Word2Vec
def tfidf_w2vec(list_of_sent):
    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 lis
    row=0;
    for sent in tqdm(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
    return tfidf_sent_vectors
```

In [53]:

```
tfidf_w2v_train = tfidf_w2vec([sent.split() for sent in train_df['CleanedText'].values])
tfidf_w2v_cv = tfidf_w2vec([sent.split() for sent in cv_df['CleanedText'].values])
tfidf_w2v_test = tfidf_w2vec([sent.split() for sent in test_df['CleanedText'].values])
```

```
100%|████████████████████████████████████████████████████████████
| 60000/60000 [30:03<00:00, 33.26it/s]
100%|████████████████████████████████████████████████████████████
| 20000/20000 [09:29<00:00, 35.09it/s]
100%|████████████████████████████████████████████████████████████
| 20000/20000 [09:04<00:00, 36.74it/s]
```

In [54]:

```
#saving tfidf weighted w2v
with open(os.path.join(dir_path, "tfidf_weighted_w2v_train.pkl"), 'wb') as tfidf_w2v_pickle
    pickle.dump(tfidf_w2v_train, tfidf_w2v_pickle)

with open(os.path.join(dir_path, "tfidf_weighted_w2v_cv.pkl"), 'wb') as tfidf_w2v_pickle:
    pickle.dump(tfidf_w2v_cv, tfidf_w2v_pickle)

with open(os.path.join(dir_path, "tfidf_weighted_w2v_test.pkl"), 'wb') as tfidf_w2v_pickle:
    pickle.dump(tfidf_w2v_test, tfidf_w2v_pickle)

print("Done!")
```

```
Done!
```

[5] KNN Assignment

In [6]:

```
# loading the libraries
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc
```

In [11]:

```
# loading labels
with open(os.path.join(dir_path, "train_lab.pkl"), 'rb') as bow:
    train_lab = pickle.load(bow)
with open(os.path.join(dir_path, "cv_lab.pkl"), 'rb') as bow:
    cv_lab = pickle.load(bow)
with open(os.path.join(dir_path, "test_lab.pkl"), 'rb') as bow:
    test_lab = pickle.load(bow)

#converting it into binary
train_lab_bin = [1 if x=='positive' else 0 for x in train_lab]
test_lab_bin = [1 if x=='positive' else 0 for x in test_lab]
cv_lab_bin = [1 if x=='positive' else 0 for x in cv_lab]
```

[5.1] KNN Brute Force

[5.1.1] Bag of Words

In [9]:

```
# loading data
with open(os.path.join(dir_path, "bow_uni_vec_train.pkl"), 'rb') as bow:
    train_data = pickle.load(bow)
with open(os.path.join(dir_path, "bow_uni_vec_cv.pkl"), 'rb') as bow:
    cv_data = pickle.load(bow)
with open(os.path.join(dir_path, "bow_uni_vec_test.pkl"), 'rb') as bow:
    test_data = pickle.load(bow)
```

In [10]:

```
for i in range(1,31,2):
    print(i, end=" ")
```

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
```

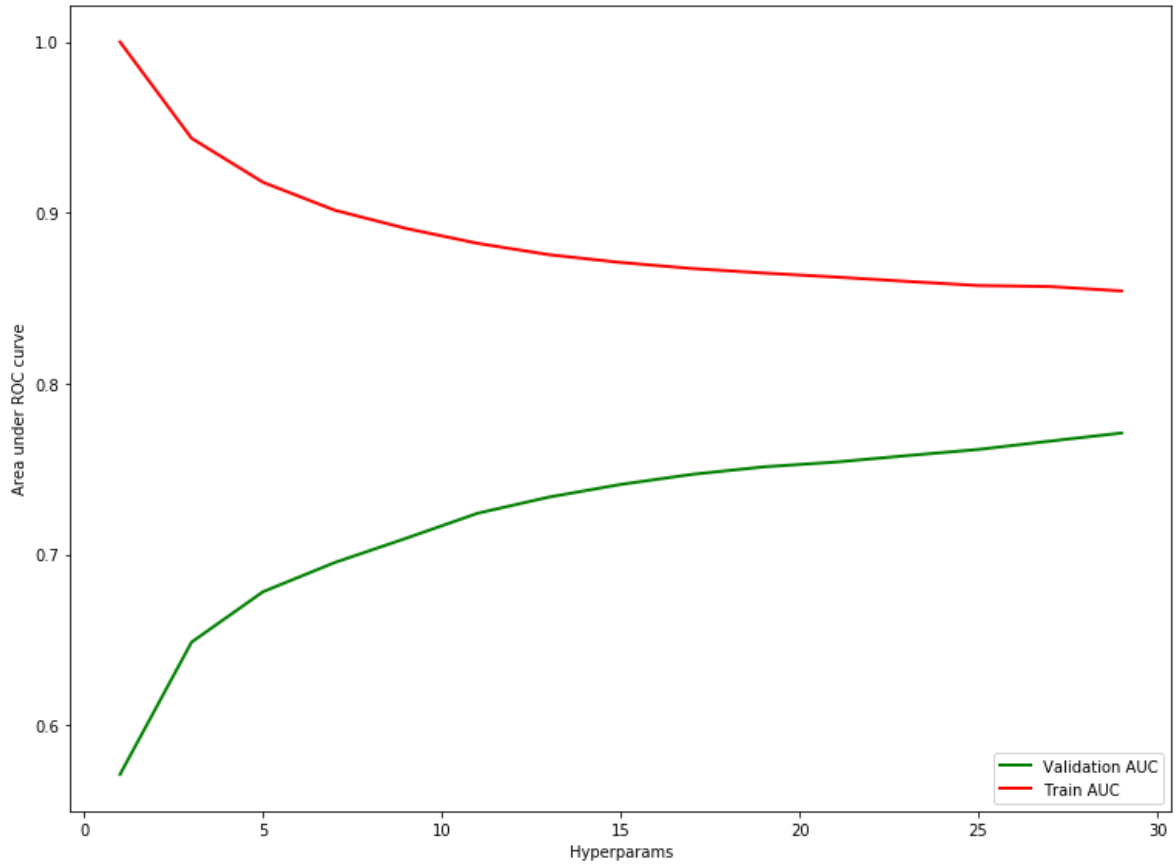
In [14]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|████████████████████████████████████████████████████████████████████
████████| 15/15 [1:37:44<00:00, 393.08s/it]
```

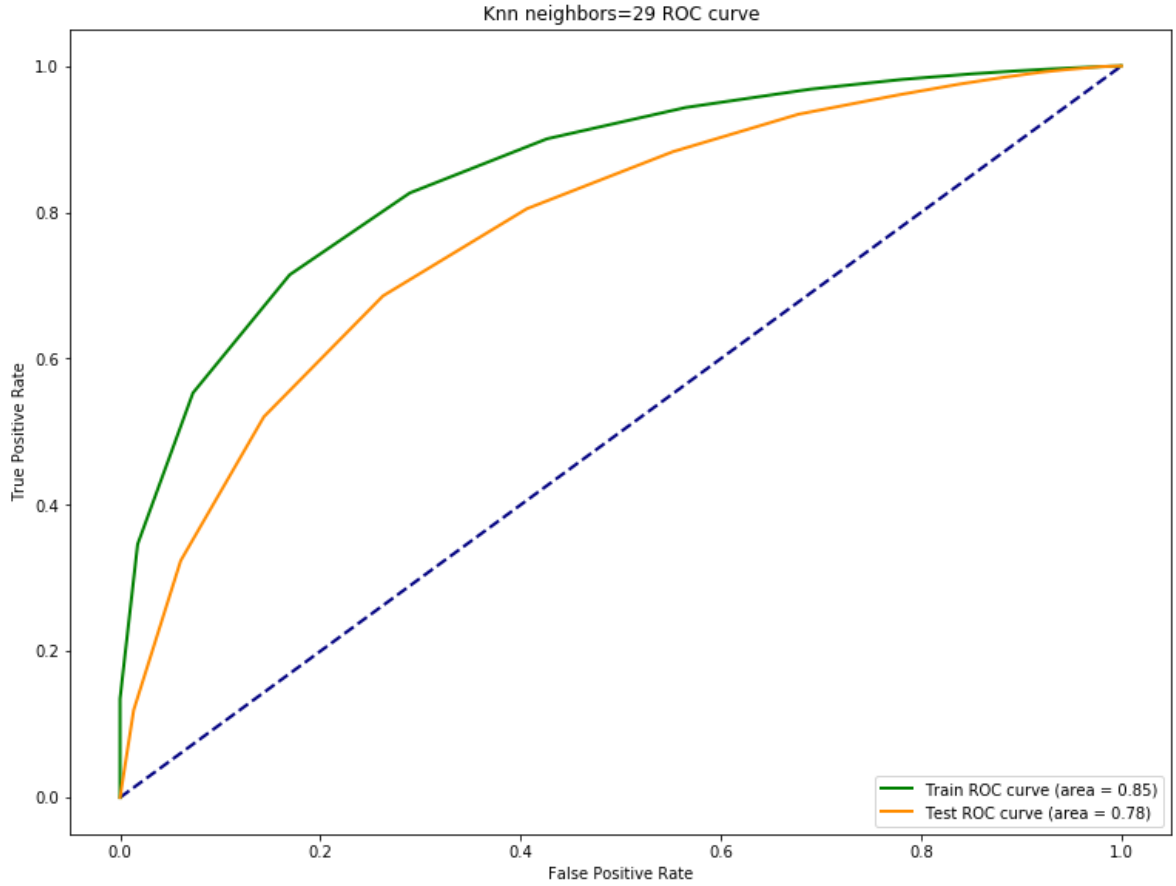
In [15]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [17]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [10]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)
```

In [17]:

```
cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)
print(cm)
print(cr)
```

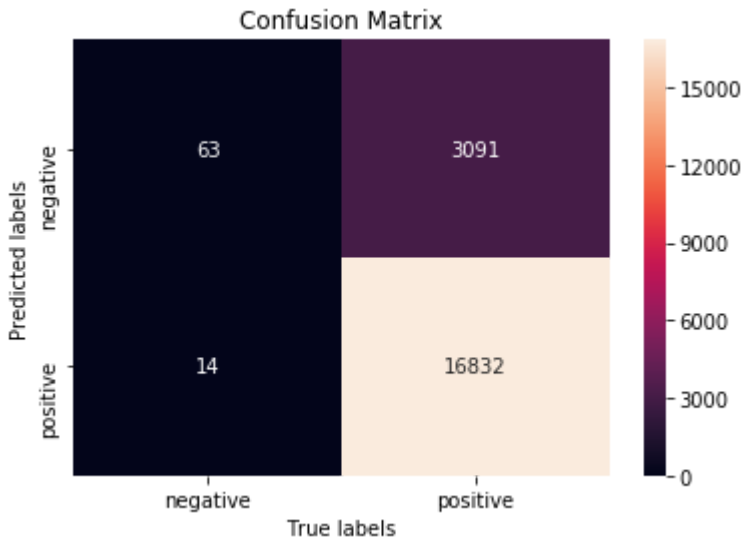
```
[[   63  3091]
 [   14 16832]]
              precision    recall  f1-score   support

           0       0.82      0.02      0.04      3154
           1       0.84      1.00      0.92     16846

   micro avg       0.84      0.84      0.84     20000
   macro avg       0.83      0.51      0.48     20000
weighted avg       0.84      0.84      0.78     20000
```

In [20]:

```
# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[20]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.1.2] TFIDF

In [21]:

```
# loading data
with open(os.path.join(dir_path, "tfidf_vec_train.pkl"), 'rb') as bow:
    train_data = pickle.load(bow)
with open(os.path.join(dir_path, "tfidf_vec_cv.pkl"), 'rb') as bow:
    cv_data = pickle.load(bow)
with open(os.path.join(dir_path, "tfidf_vec_test.pkl"), 'rb') as bow:
    test_data = pickle.load(bow)
```

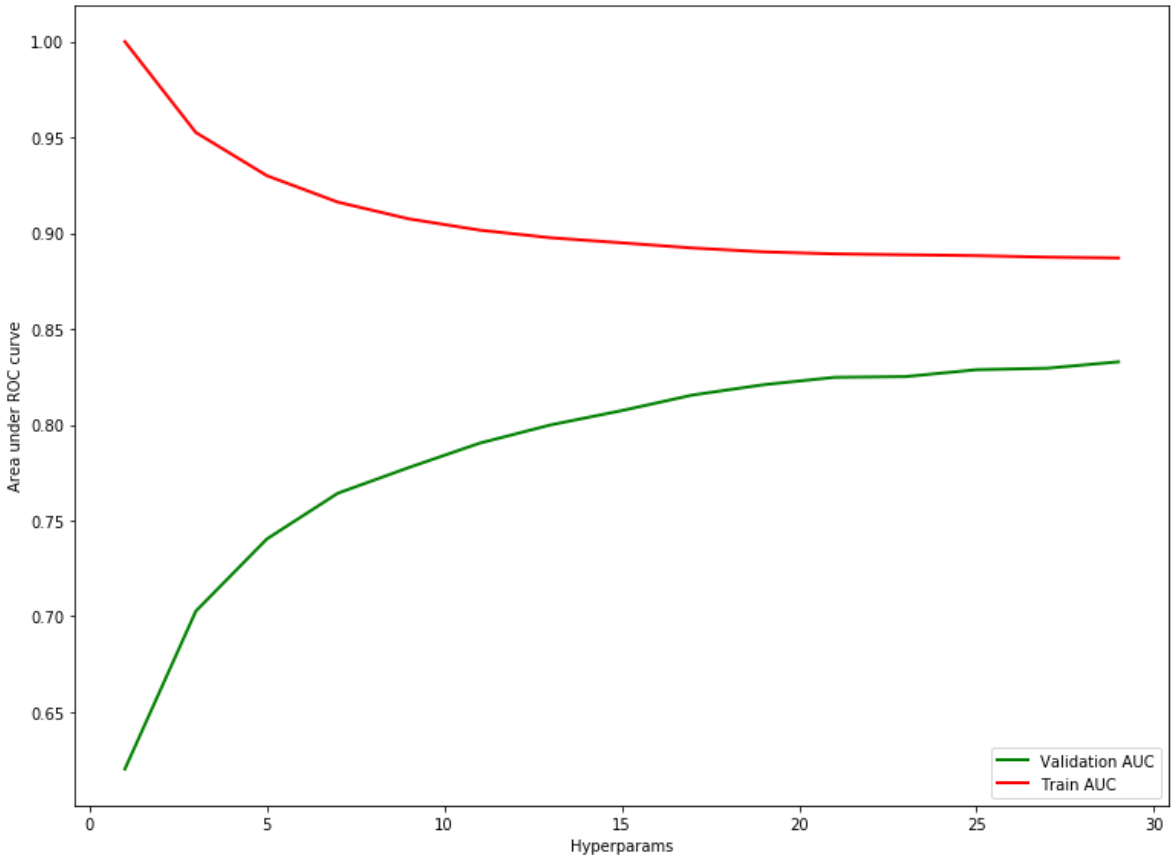
In [22]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

100%|████████| 15/15 [1:44:56<00:00, 435.54s/it]

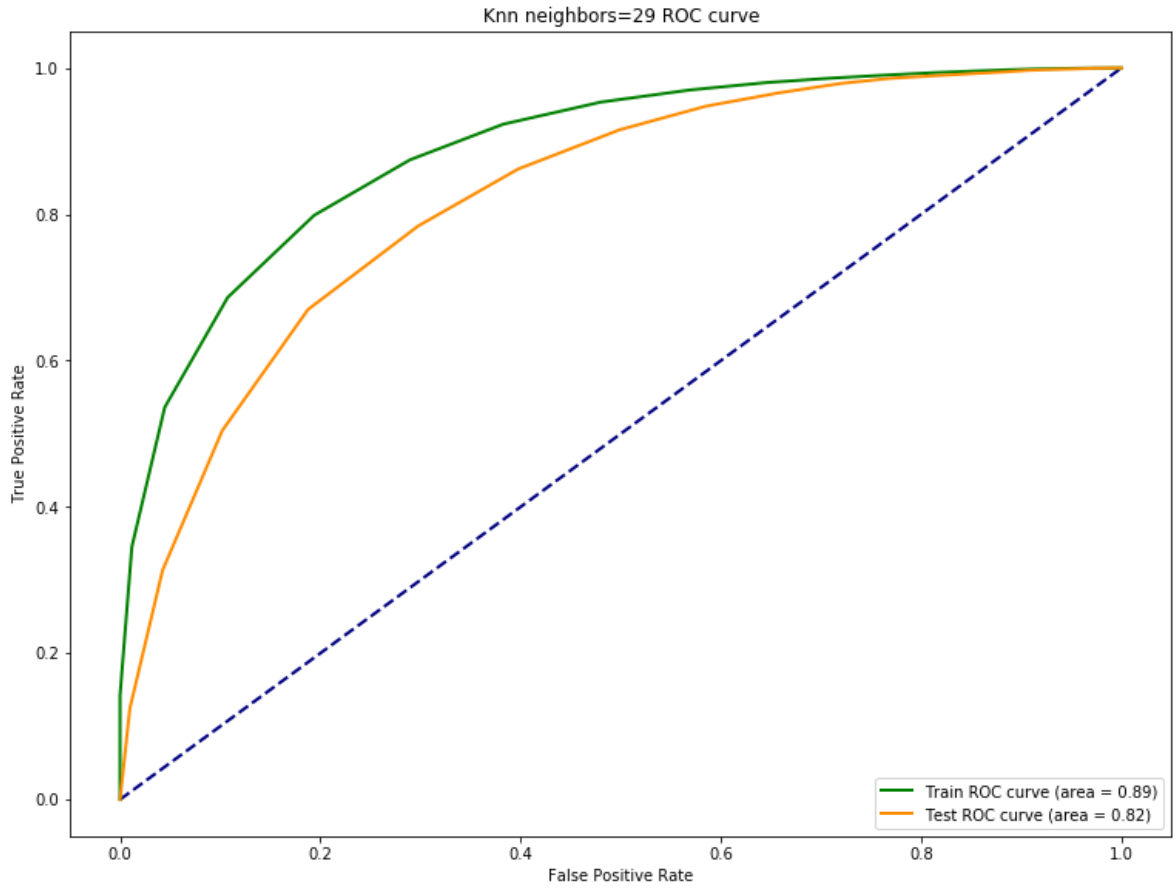
In [23]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [24]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [25]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)
```

In [26]:

```
cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)
print(cm)
print(cr)
```

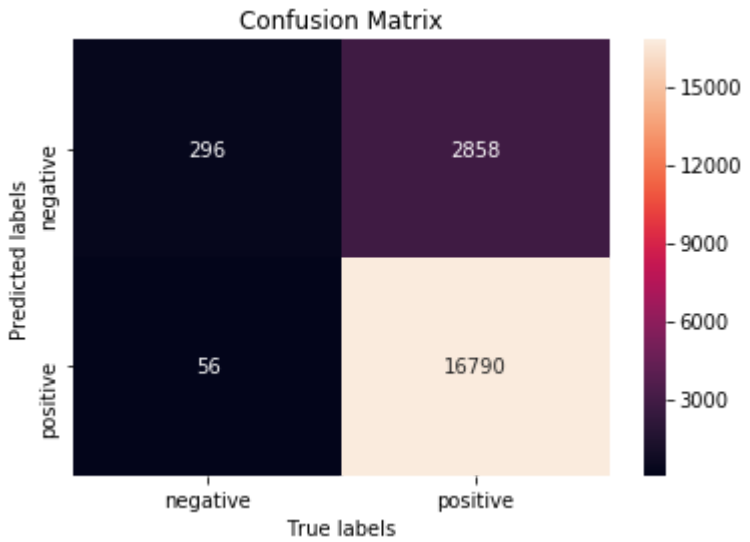
```
[[  296  2858]
 [   56 16790]]
              precision    recall  f1-score   support

           0       0.84      0.09      0.17      3154
           1       0.85      1.00      0.92     16846

   micro avg       0.85      0.85      0.85     20000
   macro avg       0.85      0.55      0.54     20000
weighted avg       0.85      0.85      0.80     20000
```

In [27]:

```
# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[27]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.1.3] Word2Vec

In [28]:

```
# loading data
with open(os.path.join(dir_path, "avg_w2v_train.pkl"), 'rb') as bow:
    train_data = pickle.load(bow)
with open(os.path.join(dir_path, "avg_w2v_cv.pkl"), 'rb') as bow:
    cv_data = pickle.load(bow)
with open(os.path.join(dir_path, "avg_w2v_test.pkl"), 'rb') as bow:
    test_data = pickle.load(bow)
```

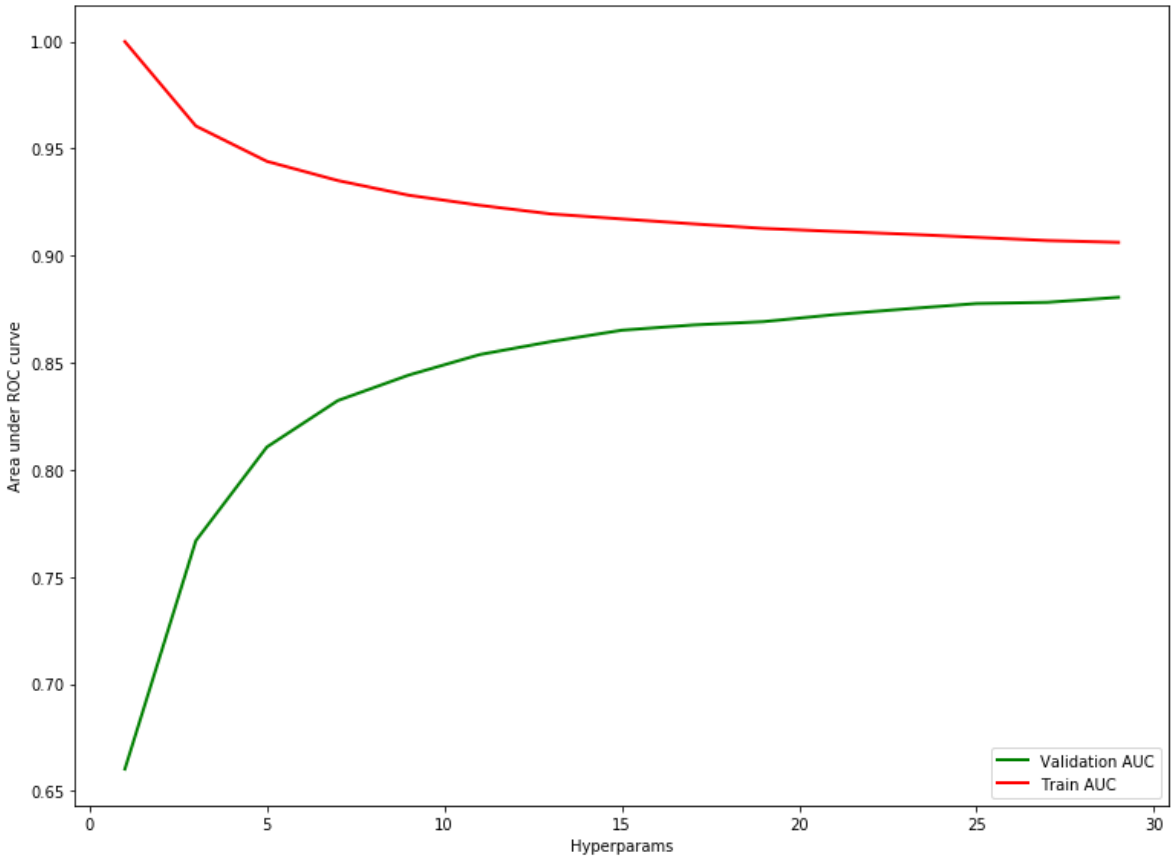
In [29]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|██████████| 15/15 [52:15<00:00, 208.51s/it]
```

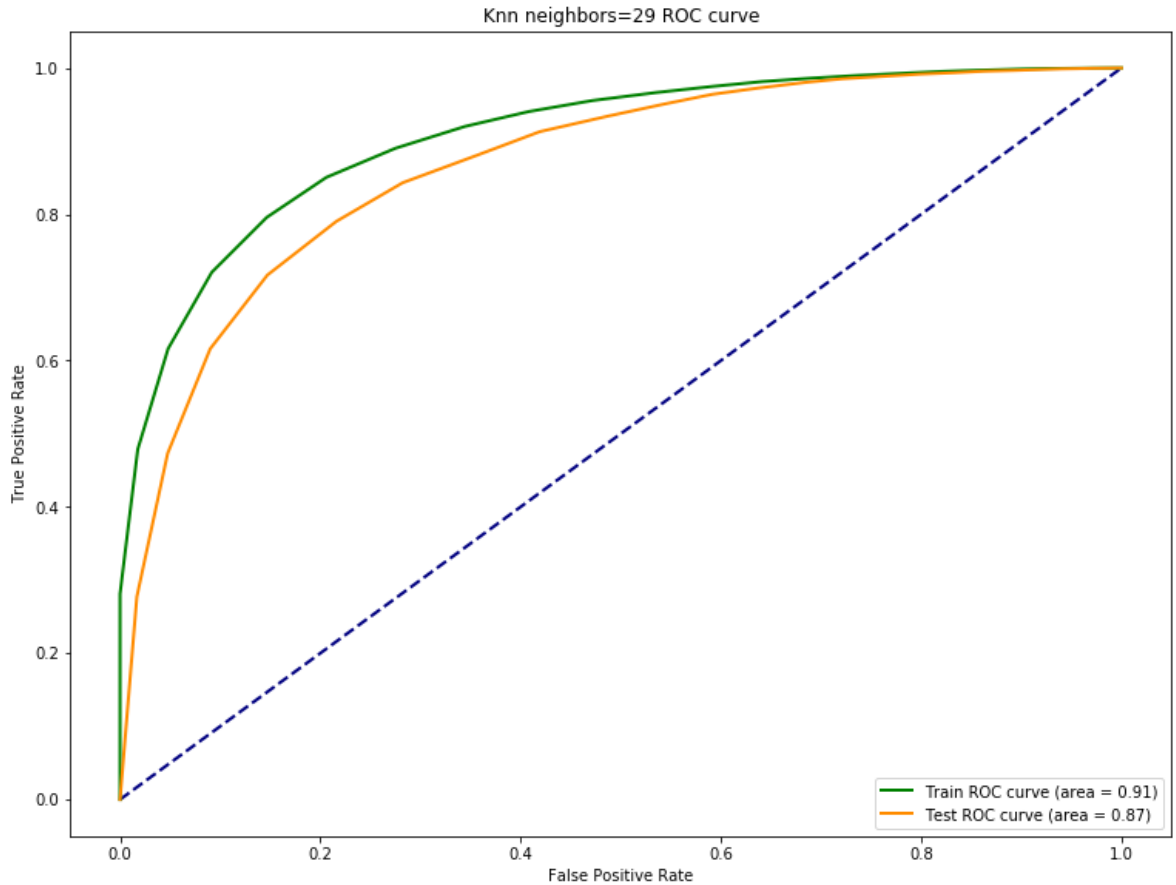
In [30]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [31]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```

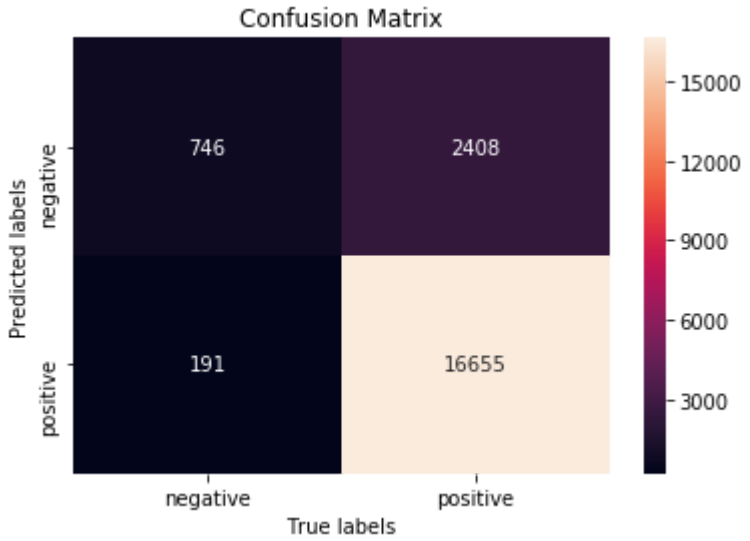


In [32]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)

cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)

# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[32]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.1.4] TFIDF Word2Vec

In [33]:

```
# loading data
with open(os.path.join(dir_path, "tfidf_weighted_w2v_train.pkl"), 'rb') as bow:
    train_data = pickle.load(bow)
with open(os.path.join(dir_path, "tfidf_weighted_w2v_cv.pkl"), 'rb') as bow:
    cv_data = pickle.load(bow)
with open(os.path.join(dir_path, "tfidf_weighted_w2v_test.pkl"), 'rb') as bow:
    test_data = pickle.load(bow)
```

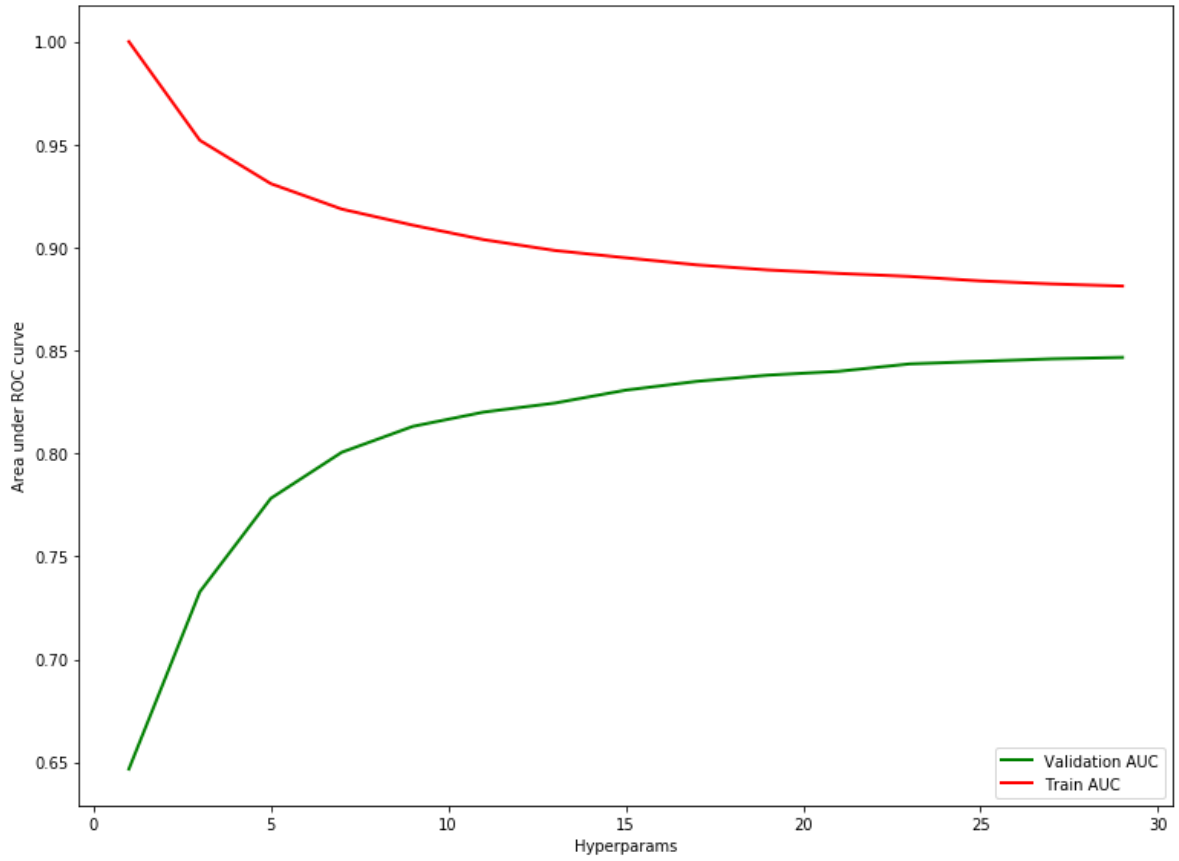
In [34]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|██████████| 15/15 [53:16<00:00, 218.84s/it]
```

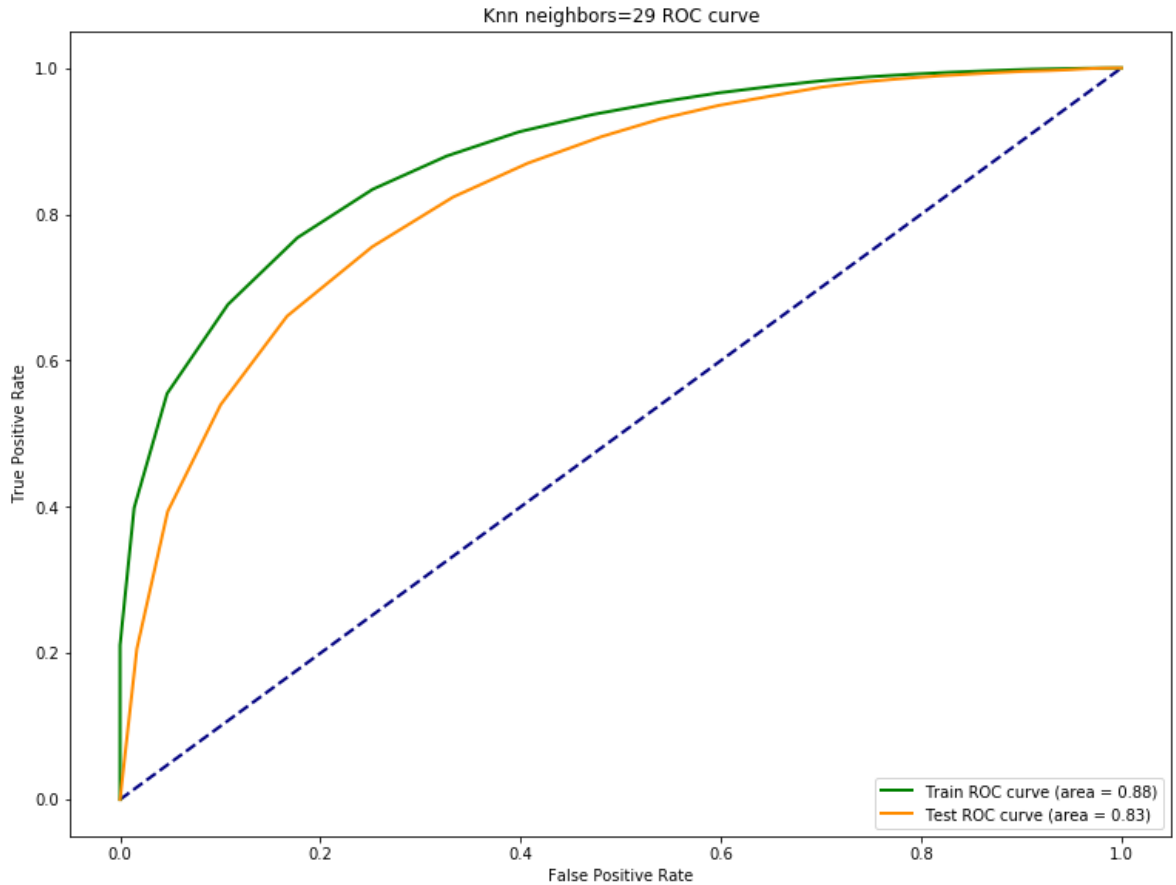
In [35]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [36]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```

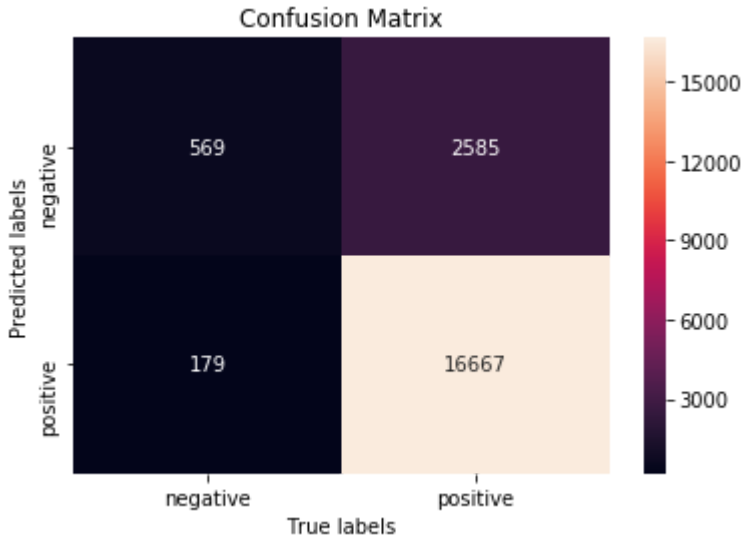


In [37]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='brute')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)

cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)

# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[37]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.2] KNN kd-tree

[5.2.1] Bag of Words

In [9]:

```
#BoW
count_vect = CountVectorizer(min_df=10, max_features=100) #in scikit-learn
train_data = count_vect.fit_transform(train_df['CleanedText'].values).toarray()
cv_data = count_vect.transform(cv_df['CleanedText'].values).toarray()
test_data = count_vect.transform(test_df['CleanedText'].values).toarray()
print("the type of count vectorizer ",type(train_data))
```

```
the type of count vectorizer  <class 'numpy.ndarray'>
```

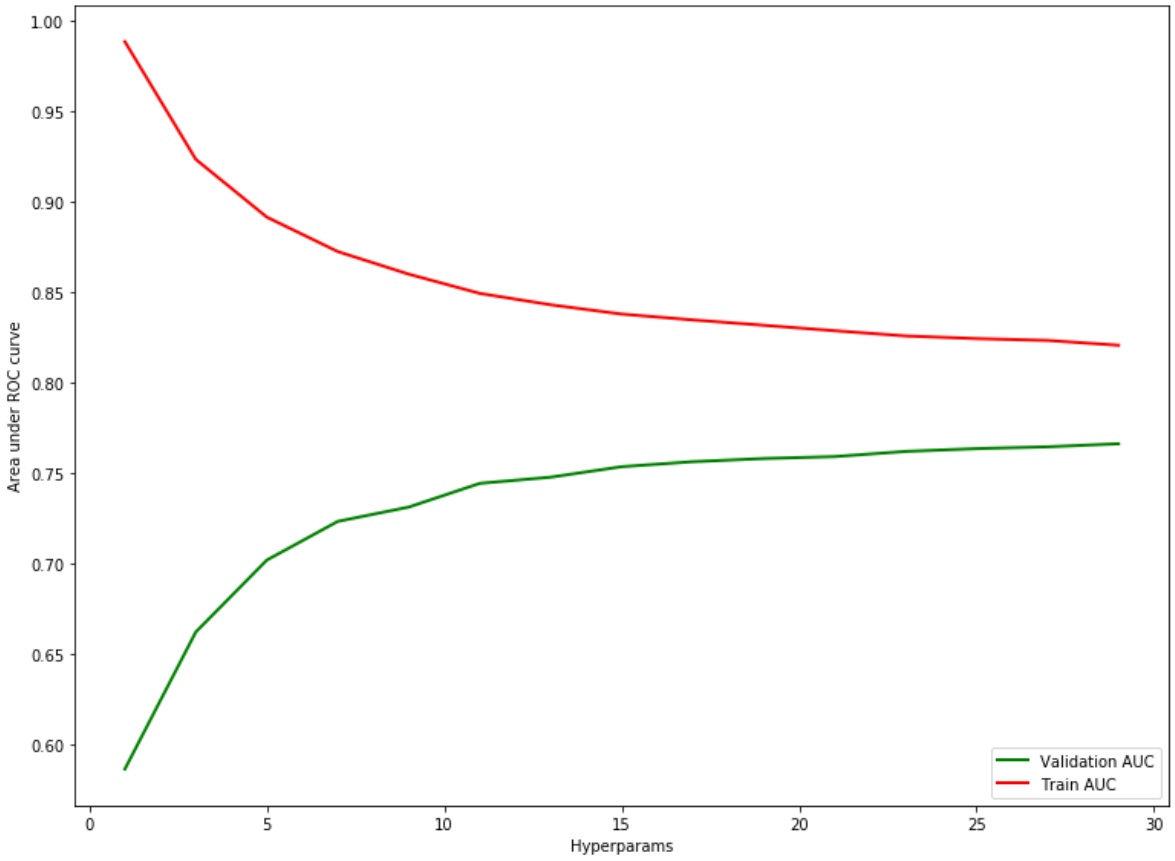
In [12]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|████████████████████████████████████████████████████████████████████
████████| 15/15 [5:45:51<00:00, 1132.54s/it]
```

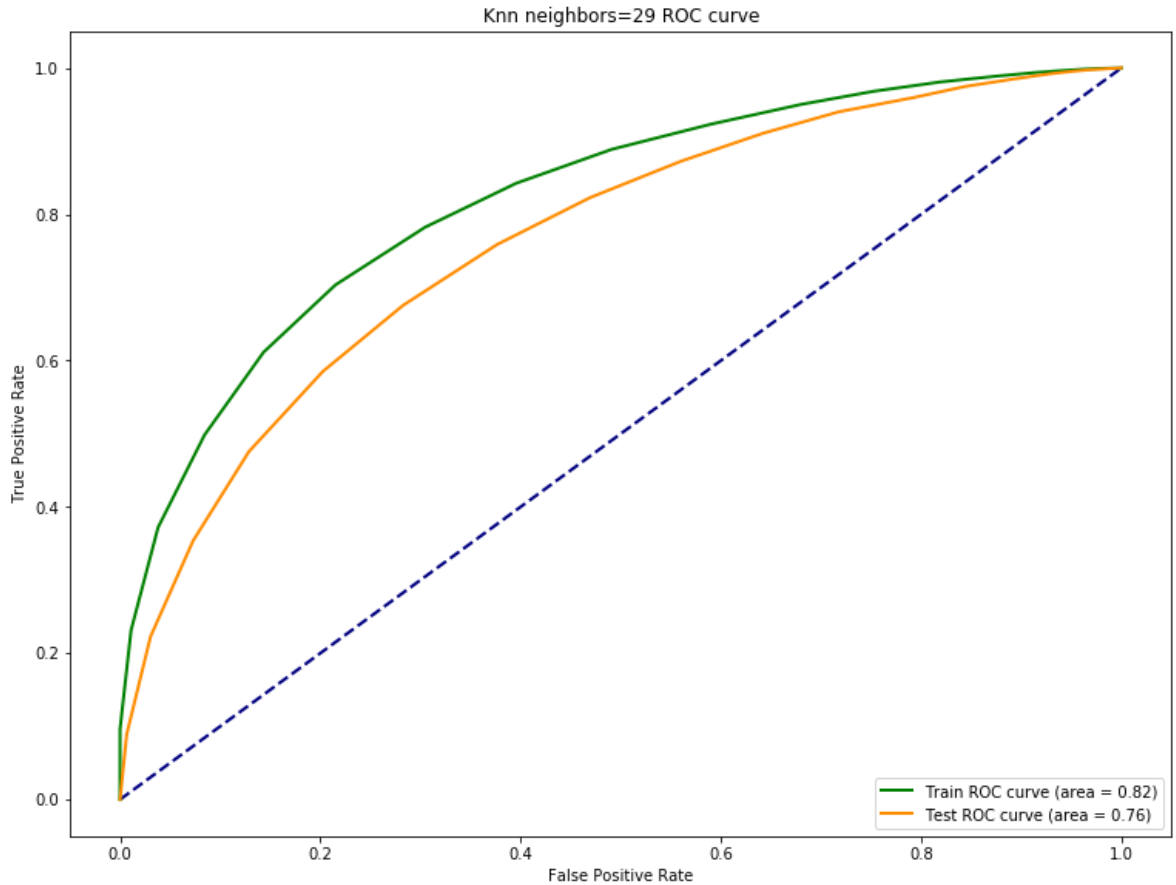
In [13]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [14]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```

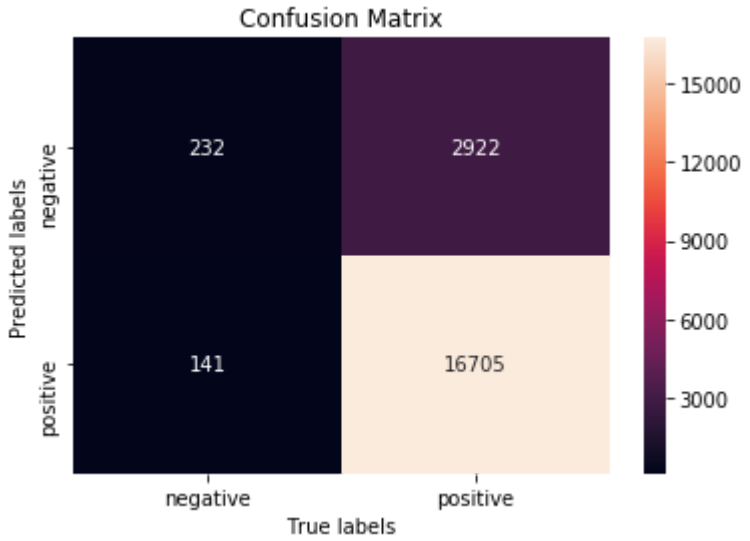


In [15]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)

cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)

# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[15]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.2.2] TFIDF

In [16]:

```
#tf-idf
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=100)
train_data = tf_idf_vect.fit_transform(train_df['CleanedText'].values).toarray()
cv_data = tf_idf_vect.transform(cv_df['CleanedText'].values).toarray()
test_data = tf_idf_vect.transform(test_df['CleanedText'].values).toarray()
print("the type of count vectorizer ",type(train_data))
```

```
the type of count vectorizer  <class 'numpy.ndarray'>
```

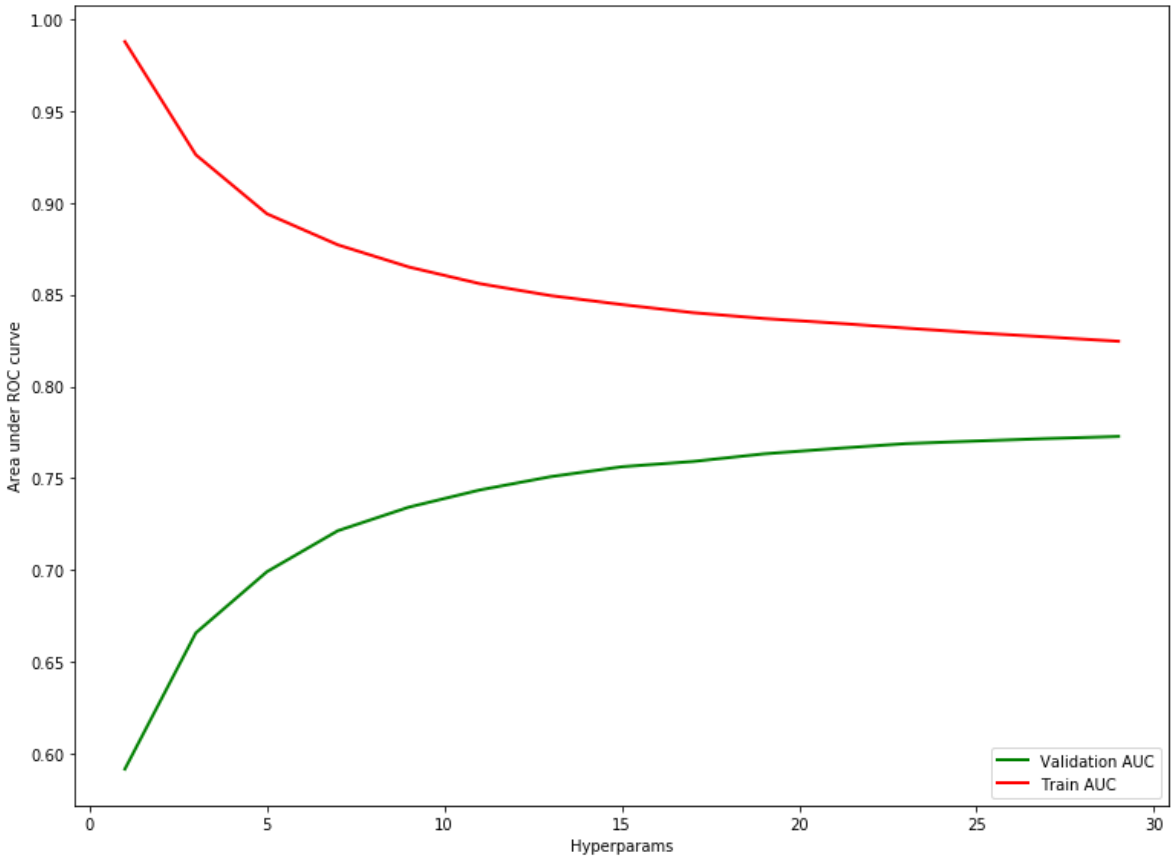
In [17]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|████████████████████████████████████████████████████████████████████
████████| 15/15 [5:41:21<00:00, 1416.00s/it]
```

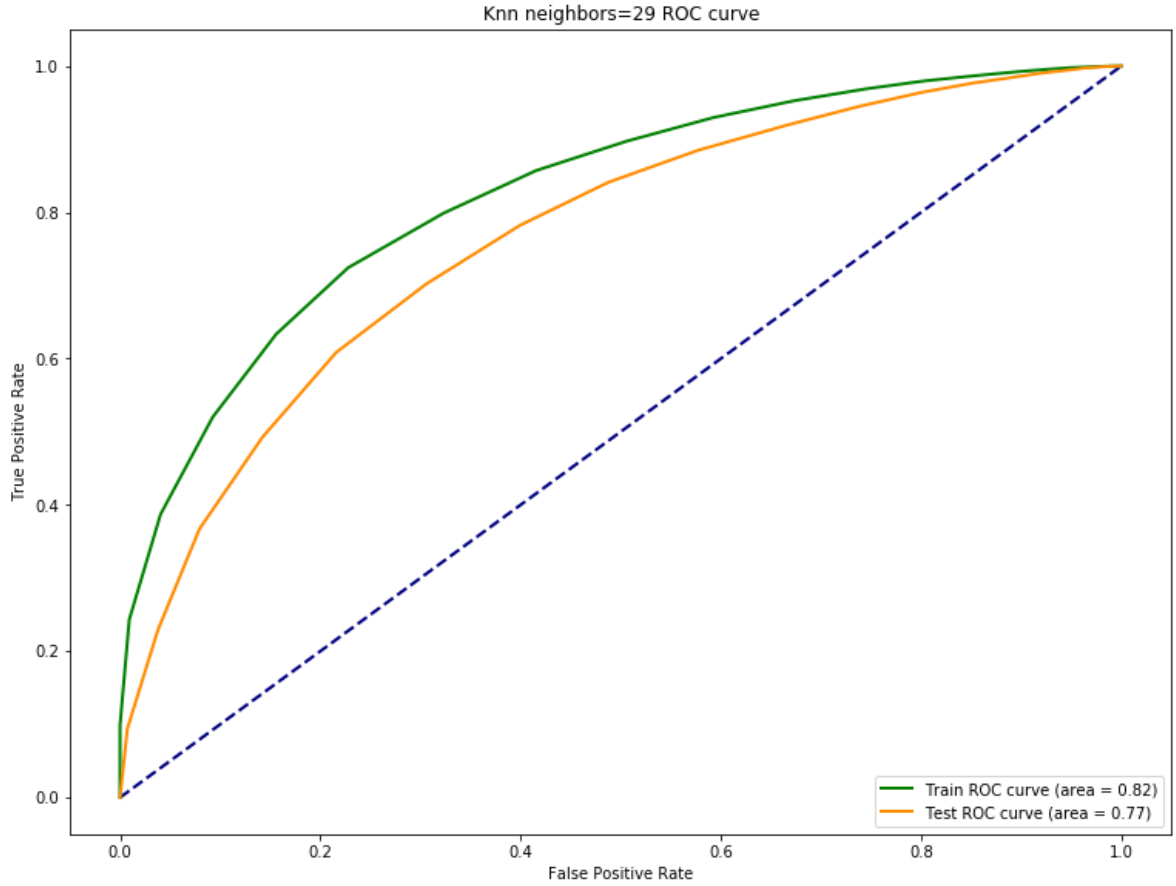
In [18]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [19]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```

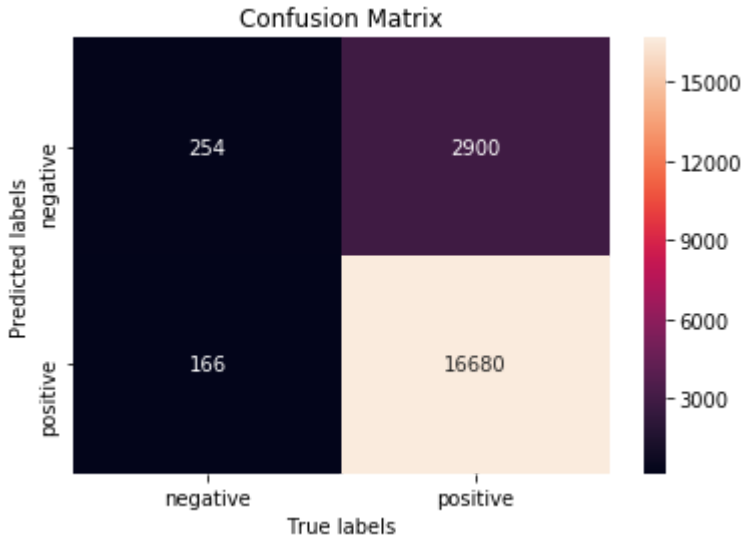


In [20]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)

cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)

# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[20]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.2.3] Word2Vec

In [21]:

```
# loading data
with open(os.path.join(dir_path, "avg_w2v_train.pkl"), 'rb') as bow:
    train_data = pickle.load(bow)
with open(os.path.join(dir_path, "avg_w2v_cv.pkl"), 'rb') as bow:
    cv_data = pickle.load(bow)
with open(os.path.join(dir_path, "avg_w2v_test.pkl"), 'rb') as bow:
    test_data = pickle.load(bow)
```

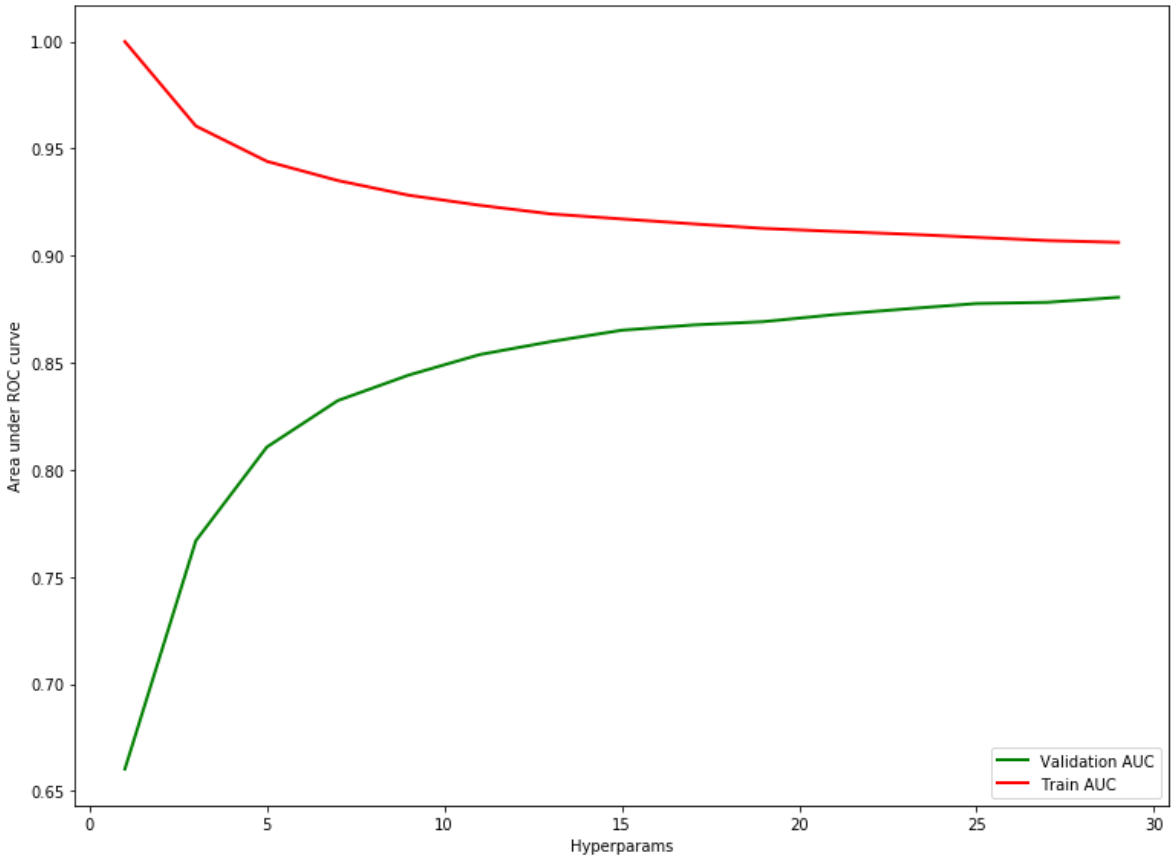
In [22]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|████████████████████████████████████████████████████████████████████
████████| 15/15 [4:38:37<00:00, 1204.26s/it]
```

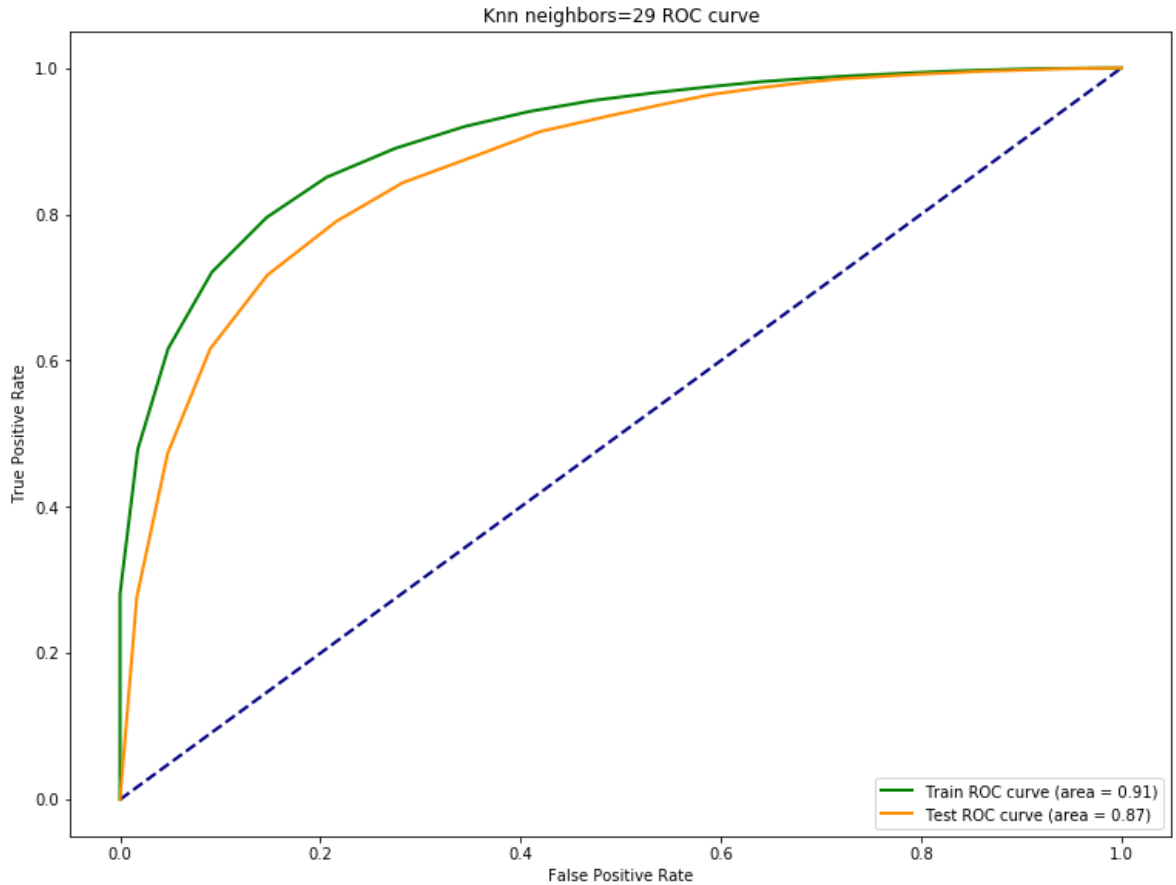
In [23]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [24]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```

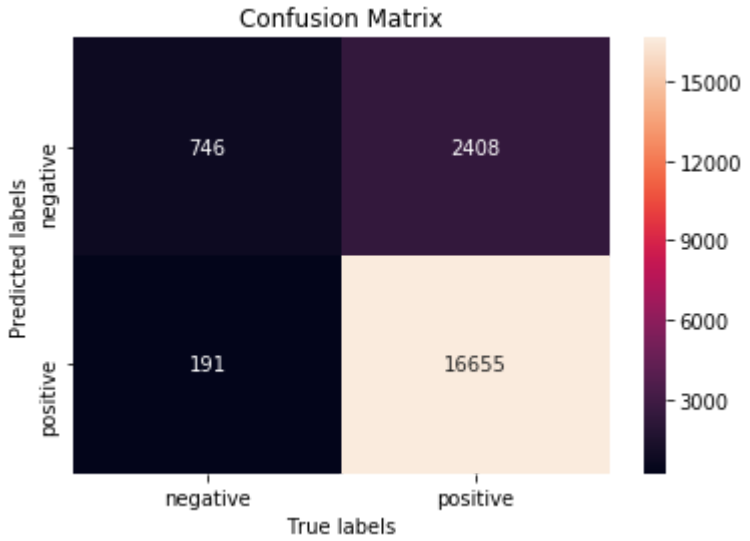


In [25]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)

cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)

# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[25]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[5.2.4] TFIDF Word2Vec

In [26]:

```
# loading data
with open(os.path.join(dir_path, "tfidf_weighted_w2v_train.pkl"), 'rb') as bow:
    train_data = pickle.load(bow)
with open(os.path.join(dir_path, "tfidf_weighted_w2v_cv.pkl"), 'rb') as bow:
    cv_data = pickle.load(bow)
with open(os.path.join(dir_path, "tfidf_weighted_w2v_test.pkl"), 'rb') as bow:
    test_data = pickle.load(bow)
```

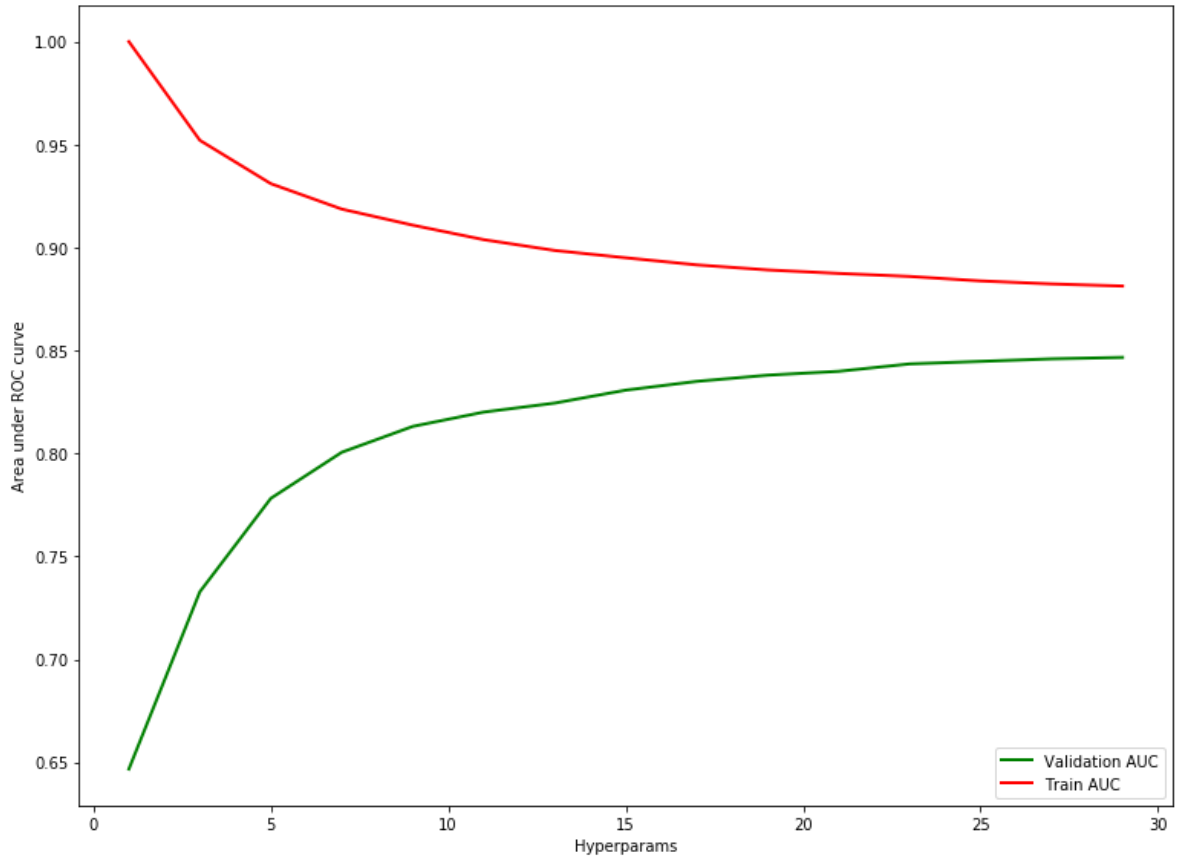
In [27]:

```
# finding best k using AUC
lw = 2
auc_train = []
auc_cv = []
auc_test = []
fpr_train = dict()
tpr_train = dict()
fpr_test = dict()
tpr_test = dict()
fpr_cv = dict()
tpr_cv = dict()

for k in tqdm(range(1, 31, 2)):
    knn_classifier = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn_classifier.fit(train_data, train_lab_bin)
    train_proba = knn_classifier.predict_proba(train_data)
    fpr_train[k], tpr_train[k], _ = roc_curve(train_lab_bin, train_proba[:,1])
    auc_train.append(auc(fpr_train[k], tpr_train[k]))

    test_proba = knn_classifier.predict_proba(test_data)
    fpr_test[k], tpr_test[k], _ = roc_curve(test_lab_bin, test_proba[:,1])
    auc_test.append(auc(fpr_test[k], tpr_test[k]))

    cv_proba = knn_classifier.predict_proba(cv_data)
    fpr_cv[k], tpr_cv[k], _ = roc_curve(cv_lab_bin, cv_proba[:,1])
    auc_cv.append(auc(fpr_cv[k], tpr_cv[k]))
```

```
100%|████████████████████████████████████████████████████████████████████
████████| 15/15 [4:07:53<00:00, 1001.95s/it]
```

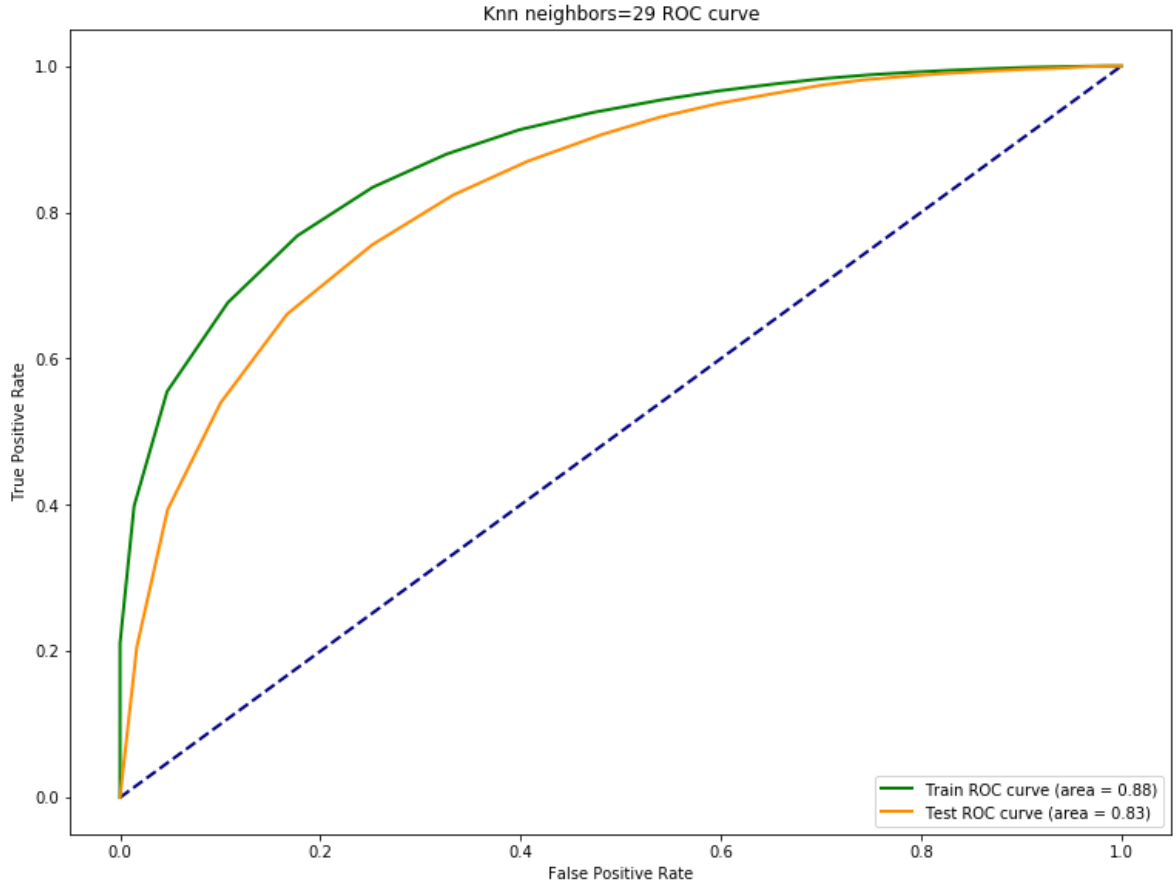
In [28]:

```
# graph train auc, cv auc and hyper params
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r

plt.figure(figsize=(12.8, 9.6))
#plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
max_idx = auc_train.index(max(auc_train))
plt.plot(range(1, 31, 2), auc_cv, color='green', lw=lw, label='Validation AUC')
plt.plot(range(1, 31, 2), auc_train, color='red', lw=lw, label='Train AUC')
plt.xlabel('Hyperparams')
plt.ylabel('Area under ROC curve')
# plt.title('Knn neighbors=' + str(max_idx+1) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [29]:

```
# plotting styles from https://scikit-learn.org/stable/auto_examples/model_selection/plot_r
plt.figure(figsize=(12.8, 9.6))
plt.plot([0,1], [0,1], color='navy', lw=lw, linestyle='--')
# calculating best k
max_idx = auc_cv.index(max(auc_cv))
max_k = 0
for idx, i in enumerate(range(1, 31, 2)):
    if idx == max_idx:
        max_k = i
        break
plt.plot(
    fpr_train[max_k], tpr_train[max_k], color='green', lw=lw,
    label='Train ROC curve (area = %0.2f)' % auc_train[max_idx]
)
plt.plot(
    fpr_test[max_k], tpr_test[max_k], color='darkorange', lw=lw,
    label='Test ROC curve (area = %0.2f)' % auc_test[max_idx]
)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn neighbors=' + str(max_k) + ' ROC curve')
plt.legend(loc="lower right")
plt.show()
```

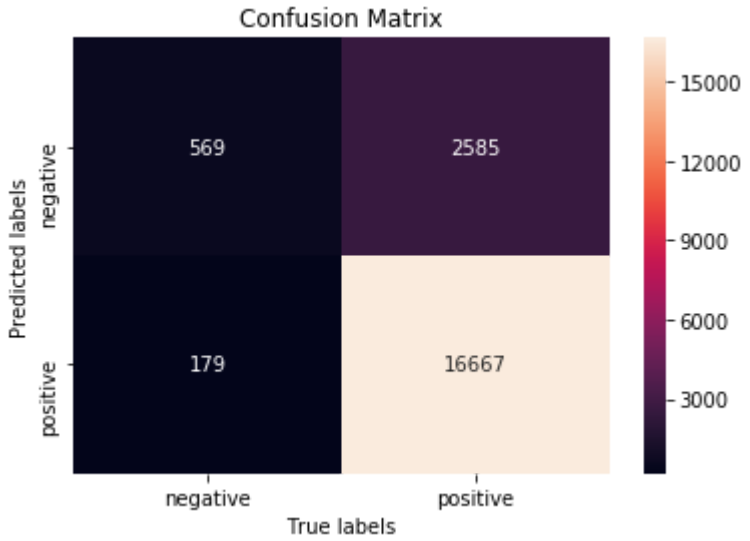


In [30]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=max_k, algorithm='kd_tree')
knn_classifier.fit(train_data, train_lab_bin)
test_predict = knn_classifier.predict(test_data)

cm = confusion_matrix(test_lab_bin, test_predict)
cr = classification_report(test_lab_bin, test_predict)

# reference https://stackoverflow.com/a/48018785
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g') #annot=True to annotate cells

# labels, title and ticks
ax.set_ylabel('Predicted labels')
ax.set_xlabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['negative', 'positive'])
ax.yaxis.set_ticklabels(['negative', 'positive'])
```

Out[30]:

```
[Text(0, 0.5, 'negative'), Text(0, 1.5, 'positive')]
```



[6] Conclusion

In [32]:

```
from prettytable import PrettyTable
```

In [33]:

```
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper parameter", "Test AUC"]
```

In [34]:

```
x.add_row(["BoW", "Brute", 29, 0.78])
x.add_row(["TFIDF", "Brute", 29, 0.82])
x.add_row(["Word2Vec", "Brute", 29, 0.87])
x.add_row(["TFIDF Word2Vec", "Brute", 27, 0.83])
x.add_row(["BoW", "kd-tree", 29, 0.76])
x.add_row(["TFIDF", "kd-tree", 29, 0.77])
x.add_row(["Word2Vec", "kd-tree", 29, 0.87])
x.add_row(["TFIDF Word2Vec", "kd-tree", 29, 0.83])
print(x)
```

```
+----------------+---------+-----------------+----------+
|   Vectorizer   |  Model  | Hyper parameter | Test AUC |
+----------------+---------+-----------------+----------+
|      BoW       |  Brute  |        29       |   0.78   |
|     TFIDF      |  Brute  |        29       |   0.82   |
|    Word2Vec    |  Brute  |        29       |   0.87   |
| TFIDF Word2Vec |  Brute  |        27       |   0.83   |
|      BoW       | kd-tree |        29       |   0.76   |
|     TFIDF      | kd-tree |        29       |   0.77   |
|    Word2Vec    | kd-tree |        29       |   0.87   |
| TFIDF Word2Vec | kd-tree |        29       |   0.83   |
+----------------+---------+-----------------+----------+
```