DecisionTree_amazon_food_review

November 17, 2018

In [3]:
```
# imported necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score, TimeSeriesSplit
#from sklearn.model_selection import cross_val_score
#from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn import model_selection
#from sklearn import cross_validation
from scipy.stats import uniform
```

In [4]:
```
import sqlite3
con = sqlite3.connect("final.sqlite")
```

In [5]:
```
cleaned_data = pd.read_sql_query("select * from Reviews", con)
```

In [6]:
```
cleaned_data.shape
```

Out[6]:
```
(364171, 12)
```

In [7]:
```
cleaned_data.head()
```

Out[7]:
```
    index      Id   ProductId          UserId                      ProfileName  \
0  138706  150524  0006641040   ACITT7DI6IDDL                  shari zychinski
1  138688  150506  0006641040  A2IW4PEEKO2R0U                            Tracy
2  138689  150507  0006641040  A1S4A3IQ2MU7V4            sally sue "sally sue"
3  138690  150508  0006641040     AZGXZ2UUK6X  Catherine Hallberg "(Kate)"
4  138691  150509  0006641040  A3CMRKGE0P909G                           Teresa

   HelpfulnessNumerator  HelpfulnessDenominator     Score        Time  \
0                     0                       0  positive   939340800
1                     1                       1  positive  1194739200
```

```
2                     1                        1  positive  1191456000
3                     1                        1  positive  1076025600
4                     3                        4  positive  1018396800

                                      Summary  \
0                    EVERY book is educational
1  Love the book, miss the hard cover version
2                chicken soup with rice months
3      a good swingy rhythm for reading aloud
4              A great way to learn the months

                                                Text  \
0  this witty little book makes my son laugh at l...
1  I grew up reading these Sendak books, and watc...
2  This is a fun way for children to learn their ...
3  This is a great little book to read aloud- it ...
4  This is a book of poetry about the months of t...

                                         CleanedText
0  b'witti littl book make son laugh loud recit c...
1  b'grew read sendak book watch realli rosi movi...
2  b'fun way children learn month year learn poem...
3  b'great littl book read nice rhythm well good ...
4  b'book poetri month year goe month cute littl ...
```

In [8]:
```python
# Sort data based on time
cleaned_data["Time"] = pd.to_datetime(cleaned_data["Time"], unit = "s")
cleaned_data = cleaned_data.sort_values(by = "Time")
cleaned_data.head()
```

Out[8]:
```
      index      Id   ProductId          UserId                 ProfileName  \
0    138706  150524  0006641040   ACITT7DI6IDDL             shari zychinski
30   138683  150501  0006641040   AJ46FKXOVC7NR          Nicholas A Mesiano
424  417839  451856  B00004CXX9   AIUWLEQ1ADEG5            Elizabeth Medina
330  346055  374359  B00004CI84  A344SMIA5JECGM             Vincent P. Ross
423  417838  451855  B00004CXX9   AJH6LUC1UT1ON  The Phantom of the Opera

     HelpfulnessNumerator  HelpfulnessDenominator     Score       Time  \
0                       0                       0  positive 1999-10-08
30                      2                       2  positive 1999-10-25
424                     0                       0  positive 1999-12-02
330                     1                       2  positive 1999-12-06
423                     0                       0  positive 2000-01-03

                                                Summary  \
0                             EVERY book is educational
30   This whole series is great way to spend time w...
424                                 Entertainingl Funny!
```

```
330                            A modern day fairy tale
423                                         FANTASTIC!

                                                  Text  \
0    this witty little book makes my son laugh at l...
30   I can remember seeing the show when it aired o...
424  Beetlejuice is a well written movie ... ever...
330  A twist of rumplestiskin captured on film, sta...
423  Beetlejuice is an excellent and funny movie. K...

                                           CleanedText
0    b'witti littl book make son laugh loud recit c...
30   b'rememb see show air televis year ago child s...
424  b'beetlejuic well written movi everyth excel a...
330  b'twist rumplestiskin captur film star michael...
423  b'beetlejuic excel funni movi keaton hilari wa...
```

In [9]:
```
cleaned_data["Score"].value_counts()
```

Out[9]:
```
positive    307061
negative     57110
Name: Score, dtype: int64
```

In [10]:
```
# Selecting top 100k data-points
final_100k = cleaned_data.iloc[:100000,:]
```

In [13]:
```
# converting scores in 0 and 1
final_100k["Score"] = final_100k["Score"].apply(lambda x: 1 if x == "positive" else 0)
```

```
C:\Users\premvardhan\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarnin
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

1 Bag of Word

In [15]:
```
# Function that will compute optimal depth for classifier using cross-validation
def tree_max_depth(X_train, y_train):

    depth_of_tree = list(range(2, 80))

    # empty list that will hold cv scores
    cv_scores = []

    # perform 10-fold cross validation
```

```
        for depth in depth_of_tree:
            tree = DecisionTreeClassifier(max_depth = depth)
            cv = TimeSeriesSplit(n_splits = 5)
            scores = cross_val_score(tree, X_train, y_train, cv = cv, scoring = 'accuracy
            cv_scores.append(scores.mean())

        # changing to misclassification error
        MSE = [1 - x for x in cv_scores]

        # determining best depth
        max_depth = depth_of_tree[MSE.index(min(MSE))]
        print('\nThe optimal depth is %d.' % max_depth)

        # plot validation error vs depth
        plt.plot(depth_of_tree, np.round(MSE, 3))
        plt.title("Validataion Error vs Depth")
        plt.xlabel('Depth')
        plt.ylabel('Validation Error')
        plt.show()

        print("The cross validation error for each depth value is : ", np.round(MSE,3))
        return max_depth
```

In [20]:
```
# 100k data which will use to train model after vectorization
X = final_100k["CleanedText"]
print("shape of X:", X.shape)
```

```
shape of X: (100000,)
```

In [21]:
```
# class label
y = final_100k["Score"]
print("shape of y:", y.shape)
```

```
shape of y: (100000,)
```

In [22]:
```
# split data into train and test where 70% data used to train model and 30% for test
from sklearn.model_selection import train_test_split
X_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_sta
print(X_train.shape, y_train.shape, x_test.shape)
```

```
(70000,) (70000,) (30000,)
```

In [23]:
```
import sklearn
print('The scikit-learn version is {}.'.format(sklearn.__version__))
```

```
The scikit-learn version is 0.20.0.
```

```
In [24]: # Train Vectorizor
         from sklearn.feature_extraction.text import CountVectorizer

         bow = CountVectorizer()
         X_train = bow.fit_transform(X_train)
         X_train
```

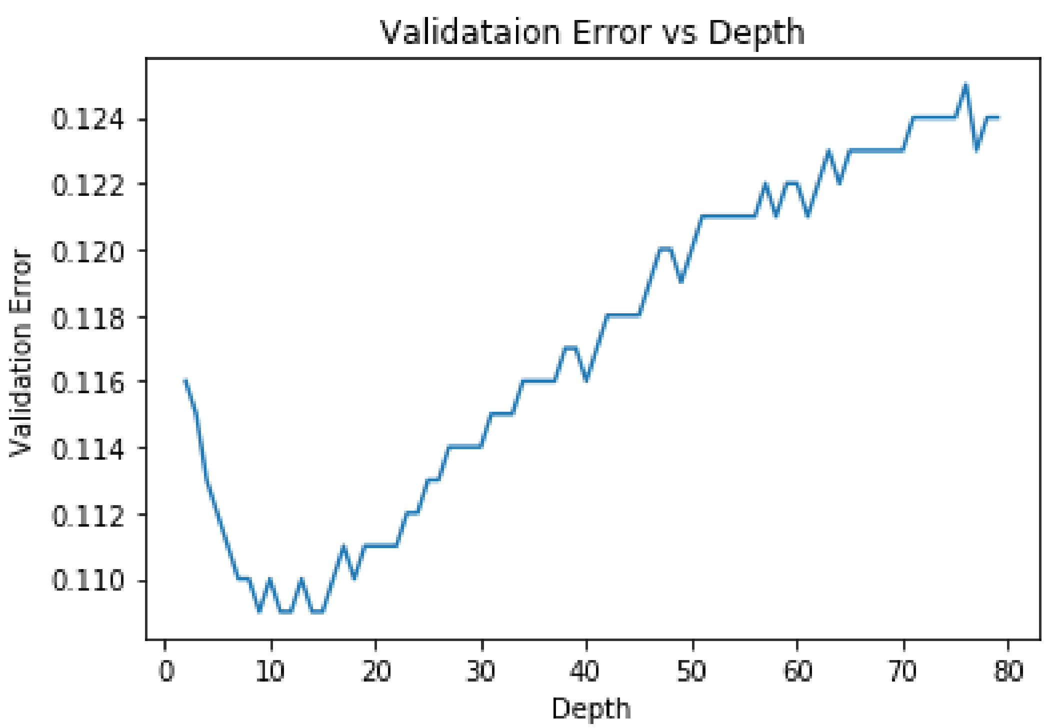
```
Out[24]: <70000x31373 sparse matrix of type '<class 'numpy.int64'>'
                 with 2094656 stored elements in Compressed Sparse Row format>
```

```
In [25]: # Test Vectorizor
         x_test = bow.transform(x_test)
         x_test
```

```
Out[25]: <30000x31373 sparse matrix of type '<class 'numpy.int64'>'
                 with 913202 stored elements in Compressed Sparse Row format>
```

```
In [26]: max_depth_bow = tree_max_depth(X_train, y_train)
         max_depth_bow
```

```
The optimal depth is 12.
```



```
The cross validation error for each depth value is :  [0.116 0.115 0.113 0.112 0.111 0.11  0.11
 0.109 0.109 0.11  0.111 0.11  0.111 0.111 0.111 0.111 0.112 0.112 0.113
 0.113 0.114 0.114 0.114 0.114 0.115 0.115 0.115 0.116 0.116 0.116 0.116
 0.117 0.117 0.116 0.117 0.118 0.118 0.118 0.118 0.119 0.12  0.12  0.119
 0.12  0.121 0.121 0.121 0.121 0.121 0.121 0.122 0.121 0.122 0.122 0.121
 0.122 0.123 0.122 0.123 0.123 0.123 0.123 0.123 0.123 0.124 0.124 0.124
 0.124 0.124 0.125 0.123 0.124 0.124]
```

```
Out[26]: 12
```

```
In [27]: # instantiate learning model max_depth = max_depth_bow
         clf = DecisionTreeClassifier(max_depth = max_depth_bow, class_weight = "balanced")
         # fitting the model
         clf.fit(X_train, y_train)
         # predict the response
         pred = clf.predict(x_test)
```

```
In [28]: train_acc_bow = clf.score(X_train, y_train)
         print("Train accuracy:",train_acc_bow)
```

```
Train accuracy: 0.7862857142857143
```

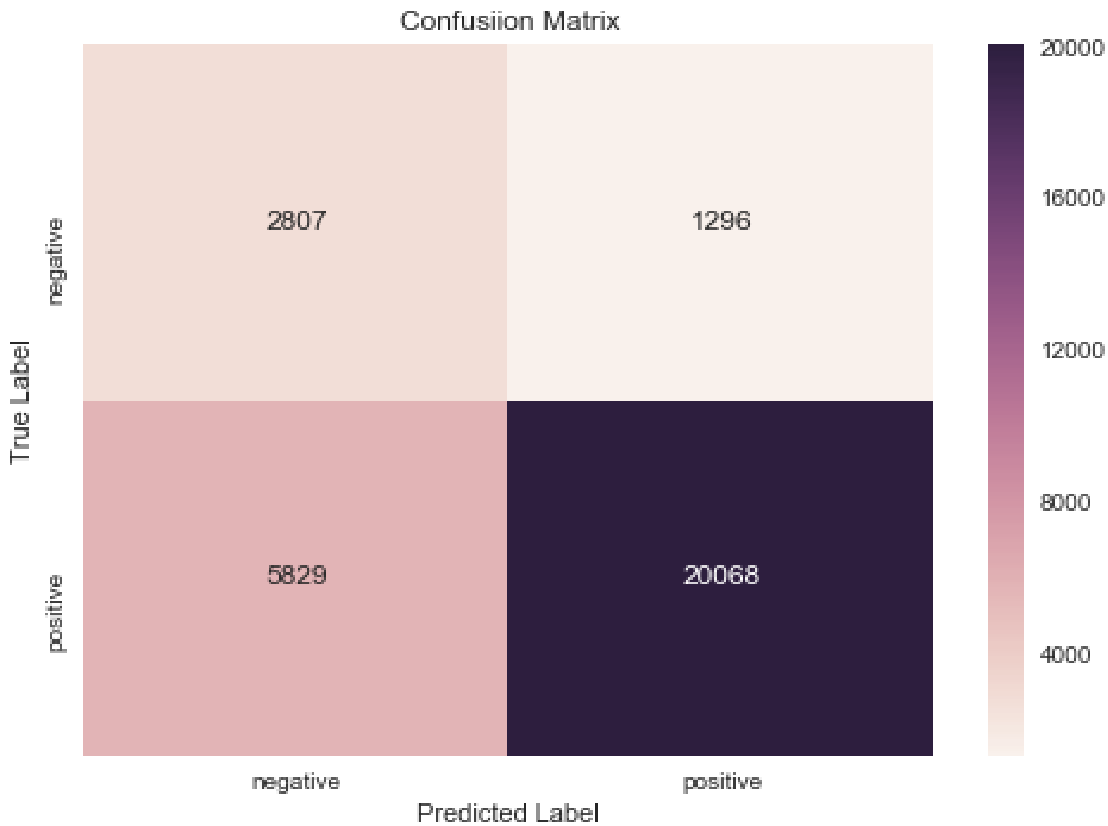
```
In [29]: test_acc_bow = accuracy_score(y_test, pred) * 100
         print('\nThe test accuracy of decision tree with depth = %f is %.2f%%' % (max_depth_bo
```

```
The test accuracy of decision tree with depth = 12.000000 is 76.25%
```

```
In [30]: # Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, pred)
         cm
```

```
Out[30]: array([[ 2807,  1296],
                [ 5829, 20068]], dtype=int64)
```

```
In [31]: # plot confusion matrix to describe the performance of classifier.
         import seaborn as sns
         class_label = ["negative", "positive"]
         df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
         sns.heatmap(df_cm, annot = True, fmt = "d")
         plt.title("Confusiion Matrix")
         plt.xlabel("Predicted Label")
         plt.ylabel("True Label")
         plt.show()
```




```
In [32]: # To show main classification report
         from sklearn.metrics import classification_report
         print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.33      0.68      0.44      4103
           1       0.94      0.77      0.85     25897

   micro avg       0.76      0.76      0.76     30000
   macro avg       0.63      0.73      0.64     30000
weighted avg       0.86      0.76      0.79     30000
```

Terminology

true positives (TP): We predicted +ve review, and review is also +ve. **true negatives (TN):** We predicted -ve, and review is also -ve. **false positives (FP):** We predicted +ve, but the review is not actually +ve.(Also known as a "Type I error.") **false negatives (FN):** We predicted -ve, but the review is actually +ve.(Also known as a "Type II error.")

Observations 1. From above figure(misclassification error vs optimal depth) It is showing that classification error for each value of depth, when depth is increaseing the error is also increasing. 2. As I tested our model on unseen data(test data) the accuracy is 76% when depth = 12. 3. In confusion matrix, It is clear that out of 30k unseen data-points classifier predict 21364 +ve and 8636 -ve class label but in real 25897 were +ve and 4103 were -ve. 4. In a nutshell we can say the generalization error is not low means this model does not works well with unseen data.

2 Tf-Idf

In [33]:
```
# data
X = final_100k["CleanedText"]
```

In [34]:
```
# Target/class-label
y = final_100k["Score"]
```

In [35]:
```
# Split data
X_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_sta
print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(70000,) (30000,) (70000,) (30000,)
```

In [36]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
X_train = tf_idf_vect.fit_transform(X_train)
X_trn = X_train
X_train
```

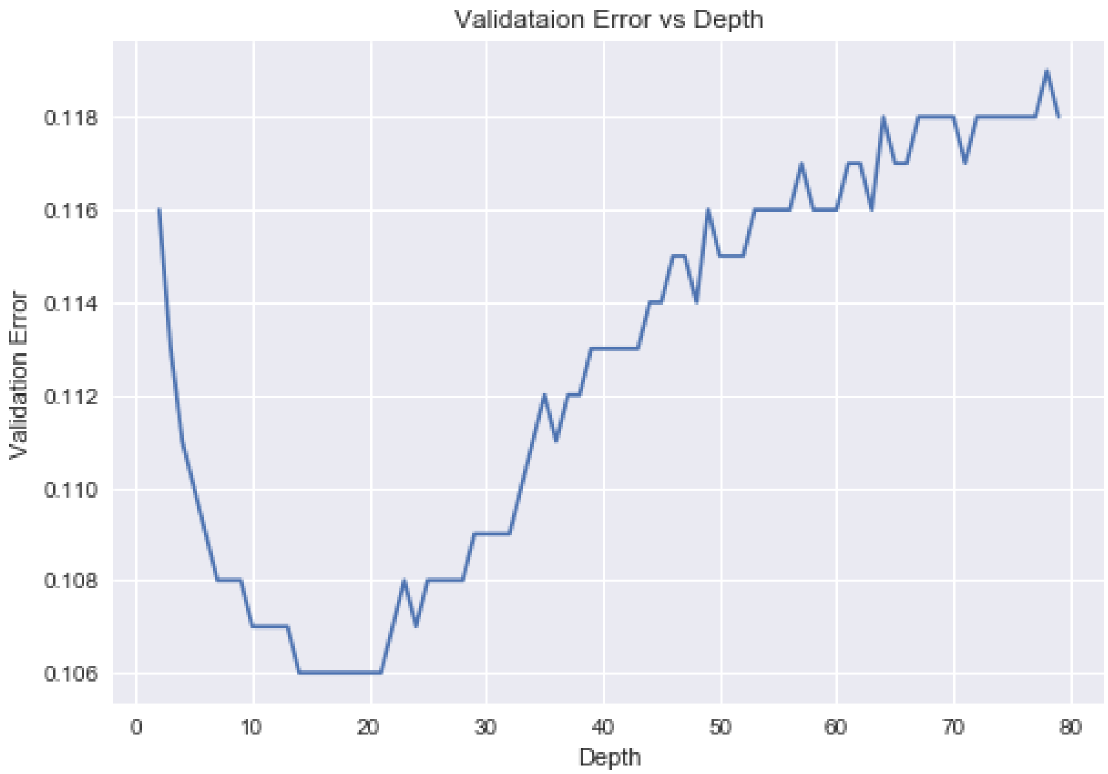
Out[36]:
```
<70000x918966 sparse matrix of type '<class 'numpy.float64'>'
        with 4504849 stored elements in Compressed Sparse Row format>
```

In [38]:
```
# Convert test text data to its vectorizor
x_test = tf_idf_vect.transform(x_test)
x_tst = x_test
x_test.shape
```

Out[38]:
```
(30000, 918966)
```

In [40]:
```
# To choose optimal_depth using cross validation
#from sklearn.model_selection import KFold
max_depth_tfidf = tree_max_depth(X_train, y_train)
max_depth_tfidf
```

```
The optimal depth is 14.
```




```
The cross validation error for each depth value is :  [0.116 0.113 0.111 0.11  0.109 0.108 0.10
 0.106 0.106 0.106 0.106 0.106 0.106 0.106 0.106 0.107 0.108 0.107 0.108
 0.108 0.108 0.108 0.109 0.109 0.109 0.109 0.11  0.111 0.112 0.111 0.112
 0.112 0.113 0.113 0.113 0.113 0.113 0.114 0.114 0.115 0.115 0.114 0.116
 0.115 0.115 0.115 0.116 0.116 0.116 0.116 0.117 0.116 0.116 0.116 0.117
 0.117 0.116 0.118 0.117 0.117 0.118 0.118 0.118 0.118 0.117 0.118 0.118
 0.118 0.118 0.118 0.118 0.119 0.118]
```

Out[40]: 14

In [41]:
```
# instantiate learning model max_depth = mas_depth_tfidf
clf = DecisionTreeClassifier(max_depth = max_depth_tfidf, class_weight = "balanced")
# fitting the model
clf.fit(X_train, y_train)
# predict the response
pred = clf.predict(x_test)
```

In [42]:
```
train_acc_tfidf = clf.score(X_train, y_train)
print("Train accuracy %f%%:" % (train_acc_tfidf))
```

```
Train accuracy 0.743043%:
```

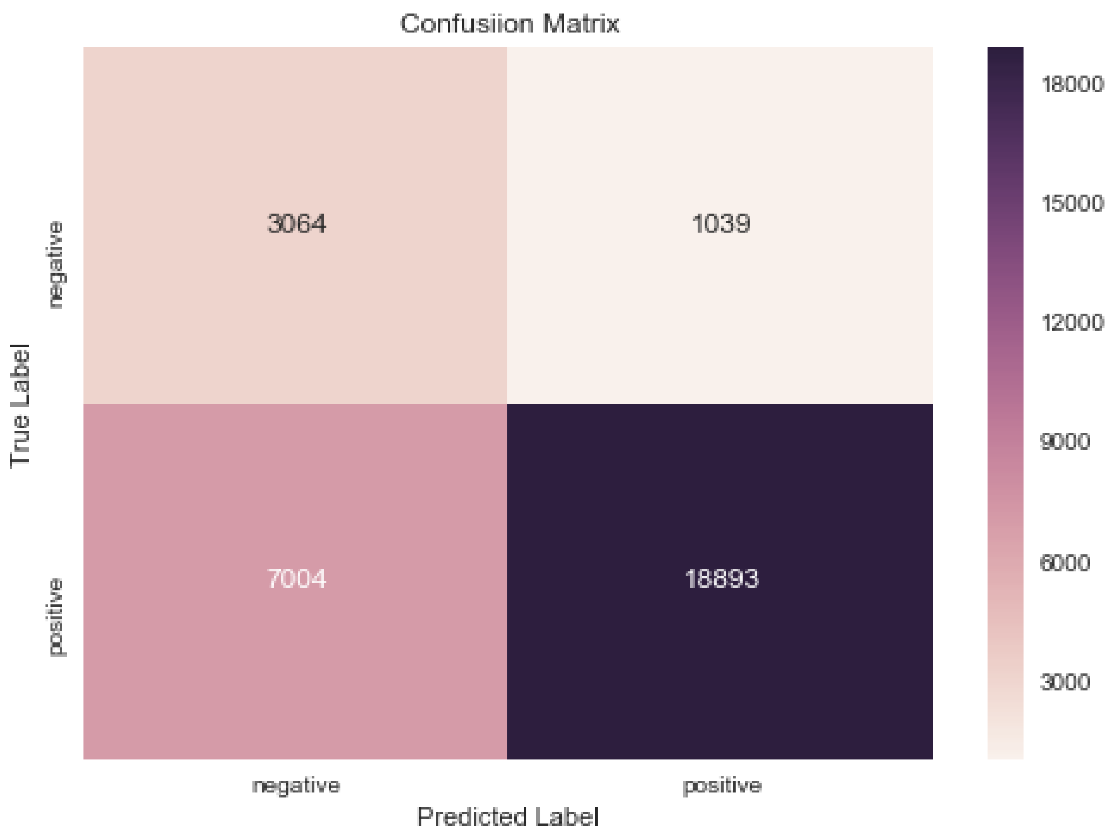
```
In [43]: test_acc_tfidf = accuracy_score(y_test, pred) * 100
         print('\nThe accuracy of the decision tree with depth = %f is %.2f%%' % (max_depth_tf
```

```
The accuracy of the decision tree with depth = 14.000000 is 73.19%
```

```
In [44]: # Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, pred)
         cm
```

```
Out[44]: array([[ 3064,  1039],
                [ 7004, 18893]], dtype=int64)
```

```
In [45]: # plot confusion matrix to describe the performance of classifier.
         import seaborn as sns
         class_label = ["negative", "positive"]
         df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
         sns.heatmap(df_cm, annot = True, fmt = "d")
         plt.title("Confusiion Matrix")
         plt.xlabel("Predicted Label")
         plt.ylabel("True Label")
         plt.show()
```



In [46]:
```
# To show main classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.30      0.75      0.43      4103
           1       0.95      0.73      0.82     25897

   micro avg       0.73      0.73      0.73     30000
   macro avg       0.63      0.74      0.63     30000
weighted avg       0.86      0.73      0.77     30000
```

Observations

1. decision tree with tfidf when depth = 14 the accuracy is low than bow.
2. In a nutshell we can say this model does not works well with unseen data.

3 Word2vec

In [133]:
```
# data
X = final_100k["Text"]
y = final_100k["Score"]
```

In [136]:
```
# Split data
X_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_sta
print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(70000,) (30000,) (70000,) (30000,)
```

In [85]:
```
import re
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 ch
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [86]:
```
# Train your own Word2Vec model using your own train text corpus
import gensim
list_of_sent=[]
```

```
#for sent in final_40k['Text'].values:
for sent in X_train:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

In [87]:
```
w2v_model_train = gensim.models.Word2Vec(list_of_sent, min_count = 5, size = 50, worke
```

In [88]:
```
w2v_model_train.wv.most_similar('like')
```

Out[88]:
```
[('think', 0.6239198446273804),
 ('prefer', 0.6220784187316895),
 ('mean', 0.5994212031364441),
 ('overwhelm', 0.593820333480835),
 ('miss', 0.5853656530380249),
 ('love', 0.578688383102417),
 ('gross', 0.5774709582328796),
 ('crave', 0.5685075521469116),
 ('awful', 0.5625168085098267),
 ('overpower', 0.5576721429824829)]
```

In [89]:
```
w2v_train = w2v_model_train[w2v_model_train.wv.vocab]
```

```
C:\Users\premvardhan\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: 
  """Entry point for launching an IPython kernel.
```

In [90]:
```
w2v_train.shape
```

Out[90]:
```
(16156, 50)
```

In [91]:
```
# Train your own Word2Vec model using your own test text corpus
import gensim
list_of_sent_test = []
#for sent in final_40k['Text'].values:
for sent in x_test:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent_test.append(filtered_sentence)
```

In [92]: w2v_model_test = gensim.models.Word2Vec(list_of_sent_test, min_count = 5, size = 50, w

In [93]: w2v_model_test.wv.most_similar('like')

Out[93]:
```
[('prefer', 0.6091516017913818),
 ('think', 0.599271297454834),
 ('know', 0.596859335899353),
 ('want', 0.5773177146911621),
 ('expect', 0.5646381378173828),
 ('miss', 0.5590541362762451),
 ('fine', 0.5548222064971924),
 ('dislike', 0.5434108972549438),
 ('enjoy', 0.5427576899528503),
 ('love', 0.5376640558242798)]
```

In [94]: w2v_test = w2v_model_test[w2v_model_test.wv.vocab]

```
C:\Users\premvardhan\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: C
  """Entry point for launching an IPython kernel.
```

In [95]: w2v_test.shape

Out[95]: (10801, 50)

4 Average word2vec

In [96]:
```
# 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
        try:
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
70000
50
```

In [97]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_test: # 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
        try:
            vec = w2v_model_test.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

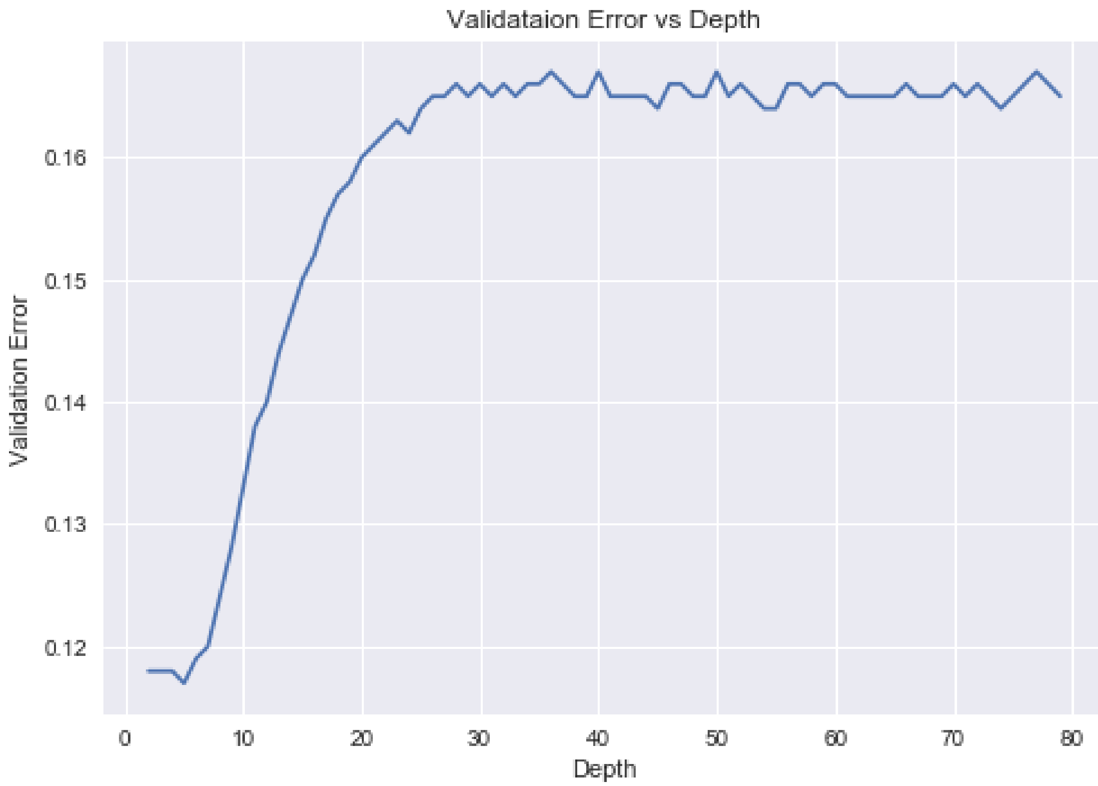
```
30000
50
```

In [98]:
```
X_train = sent_vectors
#X_train
```

In [99]:
```
x_test = sent_vectors_test
#x_test
```

In [100]:
```
# To choose best depth using cross validation
#from sklearn.model_selection import KFold
#from sklearn.model_selection import KFold
max_depth_avgw2v = tree_max_depth(X_train, y_train)
max_depth_avgw2v
```

```
The optimal depth is 5.
```




```
The cross validation error for each depth value is :  [0.118 0.118 0.118 0.117 0.119 0.12  0.12
 0.147 0.15  0.152 0.155 0.157 0.158 0.16  0.161 0.162 0.163 0.162 0.164
 0.165 0.165 0.166 0.165 0.166 0.165 0.166 0.165 0.166 0.166 0.167 0.166
 0.165 0.165 0.167 0.165 0.165 0.165 0.165 0.164 0.166 0.166 0.165 0.165
 0.167 0.165 0.166 0.165 0.164 0.164 0.166 0.166 0.165 0.166 0.166 0.165
 0.165 0.165 0.165 0.165 0.166 0.165 0.165 0.165 0.166 0.165 0.166 0.165
 0.164 0.165 0.166 0.167 0.166 0.165]
```

```
Out[100]: 5
```

```
In [101]: # instantiate learning model maz_depth = max_depth_avgw2v
          clf = DecisionTreeClassifier(max_depth = max_depth_avgw2v)
          # fitting the model
          clf.fit(X_train, y_train)
          # predict the response
          pred = clf.predict(x_test)
```

```
In [102]: train_acc_avgw2v = clf.score(X_train, y_train)
          print("Train accuracy:", train_acc_avgw2v)
```

```
Train accuracy: 0.8880571428571429
```

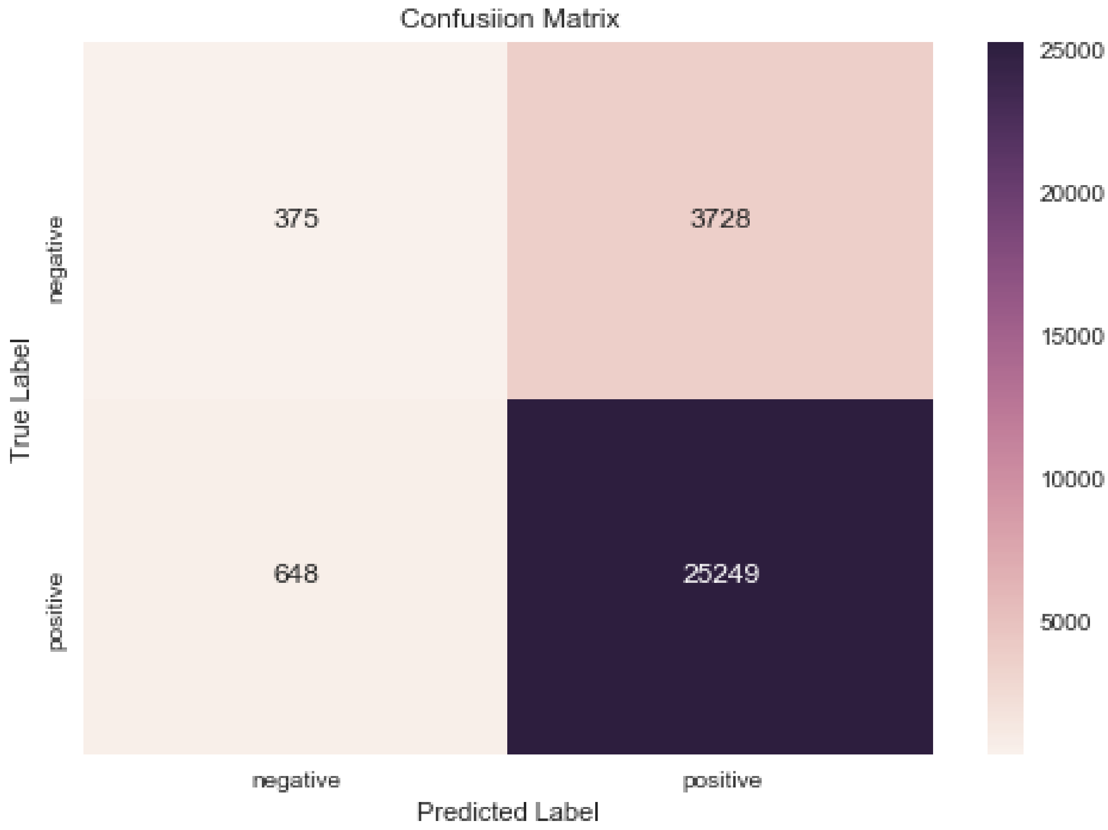
In [103]:
```
test_acc_avgw2v = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the decision tree with depth = %f is %.2f%%' % (max_depth_av
```

```
The accuracy of the decision tree with depth = 5.000000 is 85.41%
```

In [104]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, pred)
cm
```

Out[104]:
```
array([[  375,  3728],
       [  648, 25249]], dtype=int64)
```

In [105]:
```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In [106]:
```python
# To show main classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.37      0.09      0.15      4103
           1       0.87      0.97      0.92     25897

   micro avg       0.85      0.85      0.85     30000
   macro avg       0.62      0.53      0.53     30000
weighted avg       0.80      0.85      0.81     30000
```

Observations 1. Tree depth is 5 and accuracy is slightly higher than previous model.

5 TFIDF Word2Vec

In [140]:
```python
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfid

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
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
        try:
            vec = w2v_model_train.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = X_trn[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [141]:
```python
len(tfidf_sent_vectors)
```

Out[141]: 70000

In [151]:
```python
X_train = tfidf_sent_vectors
```

In [143]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfid

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in list_of_sent_test: # 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
        try:
            vec = w2v_model_test.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tfidf = x_tst[row, tfidf_feat.index(word)]
            sent_vec += (vec * tfidf)
            weight_sum += tfidf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

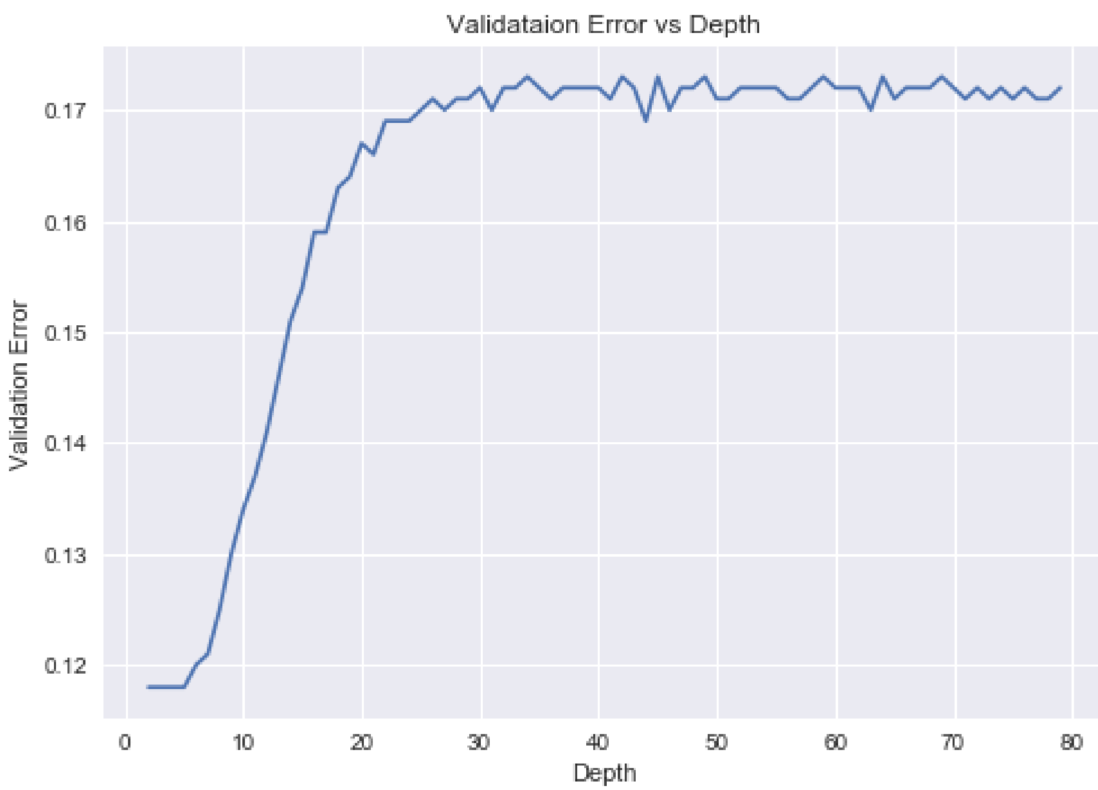
In [144]:
```
len(tfidf_sent_vectors_test)
```

Out[144]: 30000

In [147]:
```
x_test = tfidf_sent_vectors_test
```

In [154]:
```
# To choose optimal_depth using cross validation
#from sklearn.model_selection import KFold
max_depth_tfidfw2v = tree_max_depth(X_train, y_train)
max_depth_tfidfw2v
```

The optimal depth is 2.




```
The cross validation error for each depth value is :  [0.118 0.118 0.118 0.118 0.12  0.121 0.12
 0.151 0.154 0.159 0.159 0.163 0.164 0.167 0.166 0.169 0.169 0.169 0.17
 0.171 0.17  0.171 0.171 0.172 0.17  0.172 0.172 0.173 0.172 0.171 0.172
 0.172 0.172 0.172 0.171 0.173 0.172 0.169 0.173 0.17  0.172 0.172 0.173
 0.171 0.171 0.172 0.172 0.172 0.172 0.171 0.171 0.172 0.173 0.172 0.172
 0.172 0.17  0.173 0.171 0.172 0.172 0.172 0.173 0.172 0.171 0.172 0.171
 0.172 0.171 0.172 0.171 0.171 0.172]
```

Out[154]: 2

In [199]:
```
# instantiate learning model
clf = DecisionTreeClassifier(max_depth = max_depth_tfidfw2v)
# fitting the model
clf.fit(X_train, y_train)
# predict the response
pred = clf.predict(x_test)
```

Out[199]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [200]:
```
# Accuracy on train data
train_acc_tfidfw2v = clf.score(X_train, y_train)
print("Train accuracy", train_acc_tfidfw2v)
```

```
Train accuracy 0.8833142857142857
```

In [201]:
```
test_acc_tfidfw2v_grid = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the desicion tree with depth = %0.3f is %f%%' % (max_depth_
```

```
The accuracy of the desicion tree with depth = 2.000 is 86.323333%
```

In [202]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, pred)
cm
```

Out[202]:
```
array([[    0,  4103],
       [    0, 25897]], dtype=int64)
```

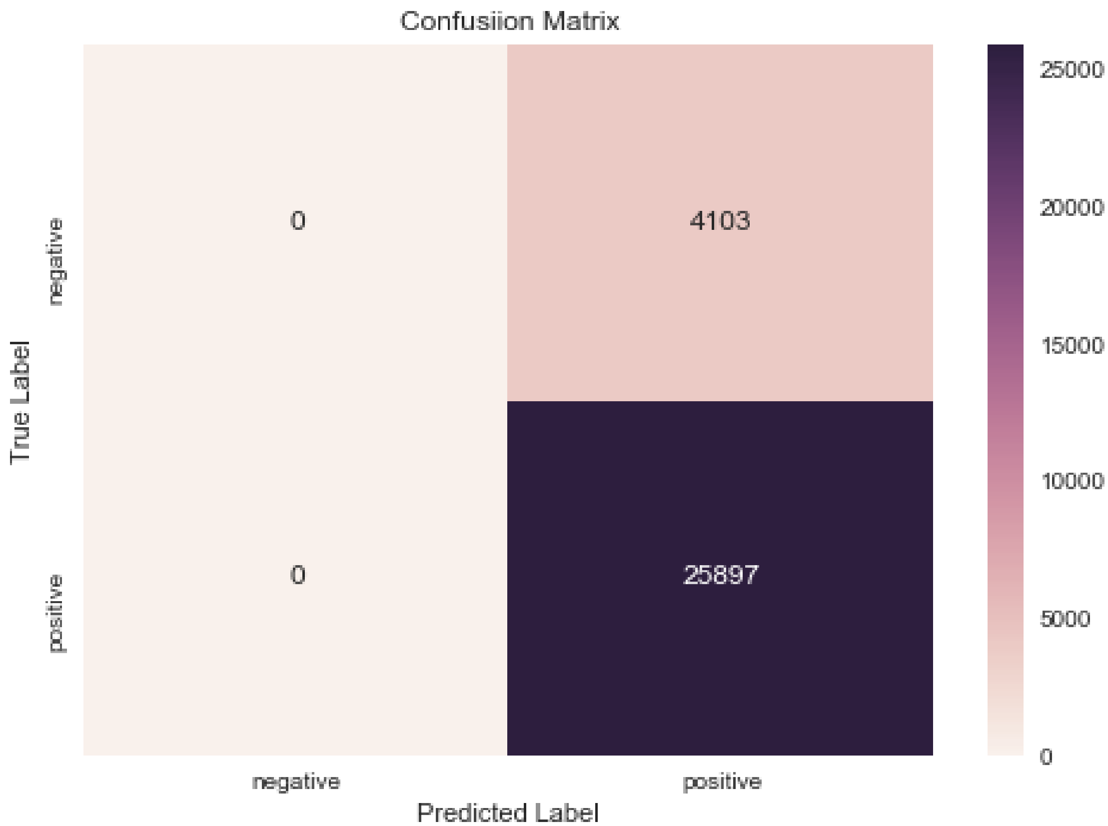
In [203]:
```
# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Out[203]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xf2b3e0fe48>
```

Out[203]:
```
<matplotlib.text.Text at 0xf2b9a69be0>
```

Out[203]:
```
<matplotlib.text.Text at 0xf2bf2845c0>
```

Out[203]:
```
<matplotlib.text.Text at 0xf2b6bef2e8>
```



In [204]:
```
# To show main classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      4103
           1       0.86      1.00      0.93     25897

   micro avg       0.86      0.86      0.86     30000
   macro avg       0.43      0.50      0.46     30000
weighted avg       0.75      0.86      0.80     30000


C:\Users\premvardhan\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: Undef
  'precision', 'predicted', average, warn_for)
```

Observations 1. The tree depth is very less and hence it could so happen that model is underfitting. 2. Confusion matrix clearing showing that our model is bias towards majority class.

Conclusions 1. Decision tree often does not work well with high dimension data. 2. As we are getting quite good accuracy for word2vec model does not mean error is less. In confusion matrix of tfidfw2v model we can see classifier is predicting every review as positive but in reality, it is not. 3. overall, None of the models are performing well on unseen data, when our performence measure is "accuracy". But, Here, We are dealing with imbalanced data, accuracy may mislead and hence we should go for f1-score, auc, balanced accuracy etc.

In [193]:
```
# model performence table
import itable
models = pd.DataFrame({'Model': ['Decision tree with Bow', "Decision tree with TFIDF"
itable.PrettyTable(models, tstyle=itable.TableStyle(theme = "theme1"), center = True
```

Out[193]:
```
<itable.itable.PrettyTable at 0xf2aff36860>
```