In [2]:
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn import cross_validation
import warnings
warnings.filterwarnings('ignore')
% matplotlib inline
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was de
precated in version 0.18 in favor of the model_selection module into which all the refactored classes and fun
ctions are moved. Also note that the interface of the new CV iterators are different from that of this modul
e. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

1.0 Data Preprocessing

In [3]:
```
# I did this on IBM cloud where file size limit was 256 MB. Hence I removed duplicates.
# Also, the score has been turned into binary class- Positive and Negative

con = sqlite3.connect(r"/resources/data/samples/Amazon_Fine_Food/final.sqlite")
filtered_data = pd.read_sql_query("""
SELECT *
FROM Reviews
""", con)
filtered_data.shape
```

Out[3]:
```
(364171, 11)
```

In [4]:
```
# Taking first 100,000 points for analysis

filtered_data = filtered_data.sort_values(by=['Time'])
final = filtered_data[:10000]
final.Score.value_counts()
```

Out[4]:
```
Positive    8868
Negative    1132
Name: Score, dtype: int64
```

In [5]:
```
import re
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import SnowballStemmer

i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        #print(i)
        #print(sent)
        break;
    i += 1;

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

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

def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/notebook/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [6]:
```
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'Positive':
                        all_positive_words.append(s) #list of all words used to describe positive reviews
                    if(final['Score'].values)[i] == 'Negative':
                        all_negative_words.append(s) #list of all words used to describe negative reviews revi
ews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

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

In [7]:
```
final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pre-processing
 of the review
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
```

In [8]:
```
final = final.sort_values(by=['Time']) #Just to be double sure that dataframe is sorted according to time

# Taking labels to make y-dimension
labels=final['Score'].values

# Checking the shape of labels
print("Shape of y-vector is",labels.shape)
```

```
Shape of y-vector is (10000,)
```

In [9]:
```
# Taking initial 70% data as training data and remaining 30% as test data

l = 0.7 * final.shape[0]
X_train = final['CleanedText'][0:int(l)]
X_test = final['CleanedText'][int(l):]
y_train = labels[0:int(l)]
y_test = labels[int(l):]
```

2.0 Bag-of-Words Model

In [20]:
```
# Making the bag of words model
# Fitting the model on Training data and transforming the test data on the fitted model
# This helps in taking care of data leakage

from sklearn.feature_extraction.text import CountVectorizer
bow = CountVectorizer()
X_train_bow = bow.fit_transform(X_train)
X_test_bow = bow.transform(X_test)
y_train_bow = y_train
y_test_bow = y_test
print("The type of count vectorizer ",type(X_train_bow))
print("The shape of count vectorizer ",X_train_bow.get_shape())
```

```
The type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of count vectorizer  (7000, 12680)
```

In [11]:
```
# Normalizing the data

from sklearn.preprocessing import StandardScaler
ss = StandardScaler(with_mean=False)
X_train_bow = ss.fit_transform(X_train_bow)
X_test_bow = ss.transform(X_test_bow)
```

In [12]:
```
# Converting 'Positive' and 'Negative' into True and False

y_train_bow = y_train_bow=='Positive'
y_test_bow = y_test_bow=='Positive'
```

Since such a dimensionality of the dataset is way too high for SVM, which is quite slow algorithm, we need to reduce the dimensionality of the dataset using TruncatedSVD.

In [13]:
```
# Making a matrix using TruncatedSVD

from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=50, n_iter=5, random_state=0)
svd.fit(X_train_bow)
print("Explained Variance = "+str(svd.explained_variance_ratio_.sum()))
```

```
Explained Variance = 0.12613330256233257
```

In [14]:
```
X_train_svd = svd.transform(X_train_bow)
X_test_svd = svd.transform(X_test_bow)
y_train_svd = y_train_bow
y_test_svd = y_test_bow
```

2.1 Grid Seach CV

In [15]:
```
# Finding the best parameters using Grid Seach CV using 10-fold Cross-Validation in Logistic Regression

from sklearn.model_selection import GridSearchCV

tuned_parameters = {'C': [10**-4 , 10**-2, 1, 100, 10000, 100000] ,
                     'gamma' : [10**-8, 10**-7,  10**-6, 10**-5, 10**-4, 10**-3, 0.01, 0.1, 1, 10, 100, 1000]
}

gridmodel = GridSearchCV(SVC(kernel = 'rbf'), tuned_parameters, scoring = 'f1_weighted', cv=3, n_jobs=-1)
gridmodel.fit(X_train_svd, y_train_svd)

print(gridmodel.best_estimator_)
print("Optimal F-score: {:.2f}".format(gridmodel.score(X_test_svd, y_test_svd)))
```

```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Optimal F-score: 0.84
```

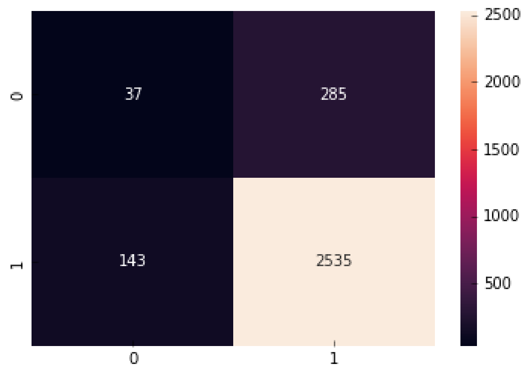
2.2 Randomized Search CV

In [19]:
```
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import RandomizedSearchCV
random_parameters =  {'C': [10**-4 , 10**-2, 1, 100, 10000, 100000] ,
                     'gamma' : [10**-8, 10**-7,  10**-6, 10**-5, 10**-4, 10**-3, 0.01, 0.1, 1, 10, 100, 1000]
}

randommodel = RandomizedSearchCV(SVC(kernel='rbf'), random_parameters, scoring = 'f1_weighted', cv=3, n_iter=2
0, n_jobs=-1)
randommodel.fit(X_train_svd, y_train_svd)

print(randommodel.best_estimator_)
print("Optimal F-score: {:.2f}".format(randommodel.score(X_test_svd, y_test_svd)))
```

```
SVC(C=100000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1000, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Optimal F-score: 0.84
```

In [21]:
```
# Fitting the best model

model = SVC(kernel='rbf', C=100, gamma=0.1)
model.fit(X_train_svd, y_train_svd)
y_pred_bow = model.predict(X_test_svd)

cm_bow=confusion_matrix(y_test_bow,y_pred_bow)
print("Confusion Matrix:")
sns.heatmap(cm_bow, annot=True, fmt='d')
plt.show()

# calculating TPR, FPR, TNR, FNR

tn, fp, fn, tp = cm_bow.ravel()

tnr_bow = tn/(tn+fp)
fpr_bow = fp/(tn+fp)
fnr_bow = fn/(fn+tp)
tpr_bow = tp/(fn+tp)

print("TPR for the model on test data is {:.2f}".format(tpr_bow))
print("FPR for the model on test data is {:.2f}".format(fpr_bow))
print("TNR for the model on test data is {:.2f}".format(tnr_bow))
print("FNR for the model on test data is {:.2f}\n".format(fnr_bow))
```

```
Confusion Matrix:
```




```
TPR for the model on test data is 0.95
FPR for the model on test data is 0.89
TNR for the model on test data is 0.11
FNR for the model on test data is 0.05
```

In [22]:
```
# calculating precision and recall

accuracy_bow = accuracy_score(y_test_bow , y_pred_bow)
precision_bow = precision_score(y_test_bow , y_pred_bow)
recall_bow = recall_score(y_test_bow , y_pred_bow)
f1_bow = f1_score(y_test_bow , y_pred_bow)

print("Accuracy score for the model on test data is {:.2f}".format(accuracy_bow))
print("Precision score for the model on test data is {:.2f}".format(precision_bow))
print("Recall score for the model on test data is {:.2f}".format(recall_bow))
print("F1 score for the model on test data is {:.2f}\n".format(f1_bow))
```

```
Accuracy score for the model on test data is 0.86
Precision score for the model on test data is 0.90
Recall score for the model on test data is 0.95
F1 score for the model on test data is 0.92
```

Hence, the positive and negative words are clearly demarcated from the logistic regression

3.0 TF-IDF Vectors

In [10]:
```
# importing the right libraries

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


# making tf-idf vector

tf_idf_vect = TfidfVectorizer() # using ngram_range as (1,1) due to computation restrictions
X_train_tf = tf_idf_vect.fit_transform(X_train)
X_test_tf = tf_idf_vect.transform(X_test)
y_train_tf = y_train
y_test_tf = y_test

print("The type of TF-IDF vectorizer ",type(X_train_tf))
print("The shape of TF-IDF vectorizer ",X_train_tf.get_shape())
```

```
The type of TF-IDF vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
The shape of TF-IDF vectorizer  (7000, 12680)
```

In [11]:
```
# Converting 'Positive' and 'Negative' into True and False

y_train_tf = y_train_tf == 'Positive'
y_test_tf = y_test_tf == 'Positive'
```

In [12]:
```
# Making a matrix using TruncatedSVD

from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=50, n_iter=5, random_state=0)
svd.fit(X_train_tf)
print("Explained Variance = "+str(svd.explained_variance_ratio_.sum()))
```

```
Explained Variance = 0.15199904880781603
```

In [13]:
```
X_train_svd = svd.transform(X_train_tf)
X_test_svd = svd.transform(X_test_tf)
y_train_svd = y_train_tf
y_test_svd = y_test_tf
```

3.1 Grid Seach CV for Optimal C and gamma

In [13]:
```
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import GridSearchCV

tuned_parameters = {'C': [10**-4 , 10**-2, 1, 100, 10000, 100000] ,
                     'gamma' : [10**-8, 10**-7,  10**-6, 10**-5, 10**-4, 10**-3, 0.01, 0.1, 1, 10, 100, 1000]
}

gridmodel = GridSearchCV(SVC(kernel = 'rbf'), tuned_parameters, scoring = 'f1_weighted', cv=3, n_jobs=-1)
gridmodel.fit(X_train_svd, y_train_svd)

print(gridmodel.best_estimator_)
print("\nOptimal F-score: {:.2f}".format(gridmodel.score(X_test_svd, y_test_svd)))
```

```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Optimal F-score: 0.88
```

3.2 Random Seach CV for Optimal λ (C) and Penalty (among L1 and L2)

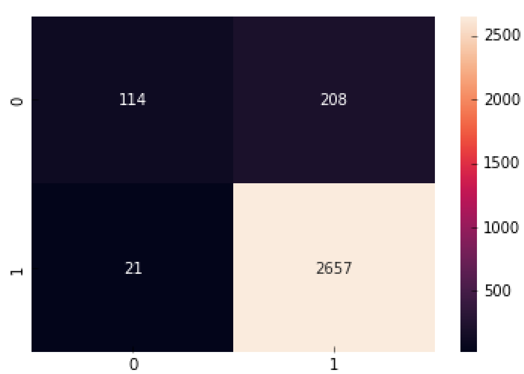
In [14]:
```
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import RandomizedSearchCV
random_parameters =  {'C': [10**-4,10**-3,10**-2,10**-1, 1, 10, 100, 1000, 10000] ,
                     'gamma' : [10**-8, 10**-7, 10**-6,10**-5,10**-4,10**-3, 0.001, 0.01, 0.1, 1, 10, 100] }

randommodel = RandomizedSearchCV(SVC(kernel='rbf'), random_parameters, scoring = 'f1_weighted', cv=3, n_iter=1
0, n_jobs=-1)
randommodel.fit(X_train_svd, y_train_svd)

print(randommodel.best_estimator_)
print("Optimal F-score: {:.2f}".format(randommodel.score(X_test_svd, y_test_svd)))
```

```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=100, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Optimal F-score: 0.85
```

3.3 Fitting the best model and calculating different performance metrics

In [15]:
```
# Fitting the best model

model = SVC(kernel='rbf', C=100, gamma=1)
model.fit(X_train_tf, y_train_tf)
y_pred_tf = model.predict(X_test_tf)

# Generating the confusion matrix
cm_tf = confusion_matrix(y_test_tf , y_pred_tf)
print("Confusion Matrix:")
sns.heatmap(cm_tf, annot=True, fmt='d')
plt.show()

# calculating TPR, FPR, TNR, FNR

tn, fp, fn, tp = cm_tf.ravel()

tnr_tf = tn/(tn+fp)
fpr_tf = fp/(tn+fp)
fnr_tf = fn/(fn+tp)
tpr_tf = tp/(fn+tp)

print("TPR for the model on test data is {:.2f}".format(tpr_tf))
print("FPR for the model on test data is {:.2f}".format(fpr_tf))
print("TNR for the model on test data is {:.2f}".format(tnr_tf))
print("FNR for the model on test data is {:.2f}\n".format(fnr_tf))
```

Confusion Matrix:




```
TPR for the model on test data is 0.99
FPR for the model on test data is 0.65
TNR for the model on test data is 0.35
FNR for the model on test data is 0.01
```

In [16]:
```
# calculating accuracy, precision and recall

accuracy_tf = accuracy_score(y_test_tf , y_pred_tf)
precision_tf = precision_score(y_test_tf , y_pred_tf)
recall_tf = recall_score(y_test_tf , y_pred_tf)
f1_tf = f1_score(y_test_tf , y_pred_tf)

print("Accuracy score for the model on test data is {:.2f}".format(accuracy_tf))
print("Precision score for the model on test data is {:.2f}".format(precision_tf))
print("Recall score for the model on test data is {:.2f}".format(recall_tf))
print("F1 score for the model on test data is {:.2f}\n".format(f1_tf))
```

```
Accuracy score for the model on test data is 0.92
Precision score for the model on test data is 0.93
Recall score for the model on test data is 0.99
F1 score for the model on test data is 0.96
```

4.0 Word2Vec

In [17]:
```
import gensim
from gensim.models import Word2Vec
```

In [18]:
```
i=0
list_of_sent=[]
for sent in X_train:
    list_of_sent.append(sent.split())

list_of_sent_tst=[]
for sent in X_test:
    list_of_sent_tst.append(sent.split())
```

In [21]:
```
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
count_vect_feat = bow.get_feature_names() # list of words in the BoW
print(count_vect_feat[count_vect_feat.index('like')])
```

```
like
```

4.1 Average Word2Vec

In [22]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
7000
50
```

In [23]:
```
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_tst: # 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_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
3000
50
```

In [24]:
```
X_train_avg = sent_vectors
X_test_avg = sent_vectors_test
y_train_avg = y_train
y_test_avg = y_test

print("Length of X_train :",len(X_train_avg))
print("Length of X_test :",len(X_test_avg))
```

```
Length of X_train : 7000
Length of X_test : 3000
```

In [25]:
```
# Converting 'Positive' and 'Negative' into True and False

y_train_avg = y_train_avg == 'Positive'
y_test_avg = y_test_avg == 'Positive'
```

4.1.1 Grid Seach CV for Optimal C and gamma

In [ ]:
```
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import GridSearchCV

tuned_parameters = {'C': [10**-4 , 10**-2, 1, 100, 10000, 100000] ,
                     'gamma' : [10**-8, 10**-7,  10**-6, 10**-5, 10**-4, 10**-3, 0.01, 0.1, 1, 10, 100, 1000]
}

gridmodel = GridSearchCV(SVC(kernel = 'rbf'), tuned_parameters, scoring = 'f1_weighted', cv=3, n_jobs=-1)
gridmodel.fit(X_train_avg, y_train_avg)

print(gridmodel.best_estimator_)
print("\nOptimal F-score: {:.2f}".format(gridmodel.score(X_test_avg, y_test_avg)))
```

```
SVC(C=10000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Optimal F-score: 0.87
```

4.1.2 Random Seach CV for C and gamma

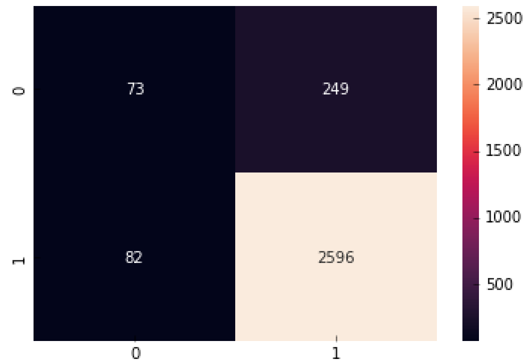
In [27]:
```python
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import RandomizedSearchCV

random_parameters =  {'C': [10**-4,10**-3,10**-2,10**-1, 1, 10, 100, 1000, 10000] ,
                     'gamma' : [10**-8, 10**-7, 10**-6,10**-5,10**-4,10**-3, 0.001, 0.01, 0.1, 1, 10, 100] }

randommodel = RandomizedSearchCV(SVC(kernel='rbf'), random_parameters, scoring = 'f1_weighted', cv=3, n_iter=1
0, n_jobs=-1)
randommodel.fit(X_train_avg, y_train_avg)

print(randommodel.best_estimator_)
print("\nOptimal F-score: {:.2f}".format(randommodel.score(X_test_avg, y_test_avg)))
```

```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Optimal F-score: 0.86
```

4.1.3 Fitting the best model and calculating different performance metrics

In [28]:
```python
# Fitting the best model

model = SVC(kernel='rbf', C=10000 , gamma=0.1)
model.fit(X_train_avg, y_train_avg)
y_pred_avg = model.predict(X_test_avg)

cm_avg = confusion_matrix(y_test_avg , y_pred_avg)
print("Confusion Matrix:")
sns.heatmap(cm_avg, annot=True, fmt='d')
plt.show()

# calculating TPR, FPR, TNR, FNR

tn, fp, fn, tp = cm_avg.ravel()

tnr_avg = tn/(tn+fp)
fpr_avg = fp/(tn+fp)
fnr_avg = fn/(fn+tp)
tpr_avg = tp/(fn+tp)

print("TPR for the model on test data is {:.2f}".format(tpr_avg))
print("FPR for the model on test data is {:.2f}".format(fpr_avg))
print("TNR for the model on test data is {:.2f}".format(tnr_avg))
print("FNR for the model on test data is {:.2f}\n".format(fnr_avg))
```

```
Confusion Matrix:
```




```
TPR for the model on test data is 0.97
FPR for the model on test data is 0.77
TNR for the model on test data is 0.23
FNR for the model on test data is 0.03
```

In [29]:
```
# calculating accuracy, precision and recall

accuracy_avg = accuracy_score(y_test_avg , y_pred_avg)
precision_avg = precision_score(y_test_avg , y_pred_avg)
recall_avg = recall_score(y_test_avg , y_pred_avg)
f1_avg = f1_score(y_test_avg , y_pred_avg)

print("Accuracy score for the model on test data is {:.2f}".format(accuracy_avg))
print("Precision score for the model on test data is {:.2f}".format(precision_avg))
print("Recall score for the model on test data is {:.2f}".format(recall_avg))
print("F1 score for the model on test data is {:.2f}\n".format(f1_avg))
```

```
Accuracy score for the model on test data is 0.89
Precision score for the model on test data is 0.91
Recall score for the model on test data is 0.97
F1 score for the model on test data is 0.94
```

4.2 TF-IDF Weighted Word2Vec

In [30]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = X_train_tf[row, tfidf_feat.index(word)]
            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

print(len(tfidf_sent_vectors))
print(len(tfidf_sent_vectors[0]))
```

```
7000
50
```

In [31]:
```
list_of_sent_tst=[]
for sent in X_test:
    list_of_sent_tst.append(sent.split())

tfidf_sent_vectors_tst=[]
row=0
for sent in list_of_sent_tst: # 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:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = X_test_tf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_tst.append(sent_vec)
    row += 1

print(len(tfidf_sent_vectors_tst))
print(len(tfidf_sent_vectors_tst[0]))
```

```
3000
50
```

In [32]:
```
X_train_w2v = tfidf_sent_vectors
X_test_w2v = tfidf_sent_vectors_tst
y_train_w2v = y_train
y_test_w2v = y_test

print("Length of X_train :",len(X_train_w2v))
print("Length of X_test :",len(X_test_w2v))
```

```
Length of X_train : 7000
Length of X_test : 3000
```

In [33]:
```
# Converting 'Positive' and 'Negative' into True and False

y_train_w2v = y_train_w2v == 'Positive'
y_test_w2v = y_test_w2v == 'Positive'
```

4.2.1 Grid Seach CV for C and gamma

In [34]:
```
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import GridSearchCV

tuned_parameters = {'C': [10**-4,10**-3,10**-2,10**-1, 1, 10, 100, 1000, 10000] ,
                     'gamma' : [10**-8, 10**-7, 10**-6,10**-5,10**-4,10**-3, 0.001, 0.01, 0.1, 1, 10, 100] }

gridmodel = GridSearchCV(SVC(kernel = 'rbf'), tuned_parameters, scoring = 'f1_weighted', cv=3, n_jobs=-1)
gridmodel.fit(X_train_w2v, y_train_w2v)

print(gridmodel.best_estimator_)
print("Optimal F-score: {:.2f}".format(gridmodel.score(X_test_w2v, y_test_w2v)))
```

```
SVC(C=10000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Optimal F-score: 0.86
```

4.1.2 Random Seach CV for Optimal λ (C) and Penalty (among L1 and L2)

In [36]:
```
# Finding the best parameters using Random Seach CV

from sklearn.model_selection import RandomizedSearchCV

random_parameters =  {'C': [10**-4,10**-3,10**-2,10**-1, 1, 10, 100, 1000, 10000] ,
                     'gamma' : [10**-8, 10**-7, 10**-6,10**-5,10**-4,10**-3, 0.001, 0.01, 0.1, 1, 10, 100] }

randommodel = RandomizedSearchCV(SVC(kernel='rbf'), random_parameters, scoring = 'f1_weighted', cv=3, n_iter=2
0, n_jobs=-1)
randommodel.fit(X_train_w2v, y_train_w2v)

print(randommodel.best_estimator_)
print("\nOptimal F-score: {:.2f}".format(randommodel.score(X_test_w2v, y_test_w2v)))
```

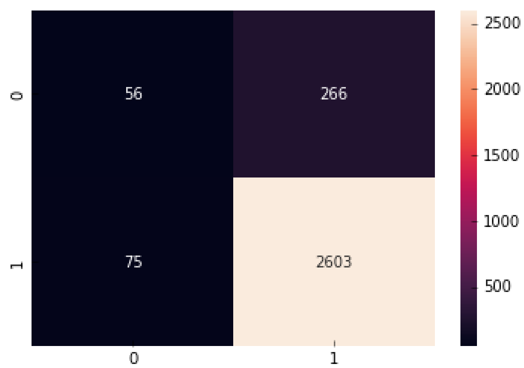
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=10, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Optimal F-score: 0.85
```

Here we got exactly the same value of F-Score everytime, both for L1 and L2 regularization. We will take optimal model as L2 regularization for C=10000

4.2.3 Fitting the best model and calculating different performance metrics

In [37]:
```python
# Fitting the best model

model = SVC(kernel='rbf', C=10000 , gamma=0.1)
model.fit(X_train_w2v, y_train_w2v)
y_pred_w2v = model.predict(X_test_w2v)

# Generating the confusion matrix
cm_w2v = confusion_matrix(y_test_w2v , y_pred_w2v)
print("Confusion Matrix:")
sns.heatmap(cm_w2v, annot=True, fmt='d')
plt.show()

# calculating TPR, FPR, TNR, FNR

tn, fp, fn, tp = cm_w2v.ravel()

tnr_w2v = tn/(tn+fp)
fpr_w2v = fp/(tn+fp)
fnr_w2v = fn/(fn+tp)
tpr_w2v = tp/(fn+tp)

print("TPR for the model on test data is {:.2f}".format(tpr_w2v))
print("FPR for the model on test data is {:.2f}".format(fpr_w2v))
print("TNR for the model on test data is {:.2f}".format(tnr_w2v))
print("FNR for the model on test data is {:.2f}\n".format(fnr_w2v))
```

```
Confusion Matrix:
```




```
TPR for the model on test data is 0.97
FPR for the model on test data is 0.83
TNR for the model on test data is 0.17
FNR for the model on test data is 0.03
```

In [38]:
```python
# calculating accuracy, precision and recall

accuracy_w2v = accuracy_score(y_test_w2v , y_pred_w2v)
precision_w2v = precision_score(y_test_w2v , y_pred_w2v)
recall_w2v = recall_score(y_test_w2v , y_pred_w2v)
f1_w2v = f1_score(y_test_w2v , y_pred_w2v)

print("Accuracy score for the model on test data is {:.2f}".format(accuracy_w2v))
print("Precision score for the model on test data is {:.2f}".format(precision_w2v))
print("Recall score for the model on test data is {:.2f}".format(recall_w2v))
print("F1 score for the model on test data is {:.2f}\n".format(f1_w2v))
```

```
Accuracy score for the model on test data is 0.89
Precision score for the model on test data is 0.91
Recall score for the model on test data is 0.97
F1 score for the model on test data is 0.94
```

5.0 Conclusion

RBF-SVM was successfully applied on all the text-processing techniques.

In order to accomplish this task, 4 types of Text processing techniques were applied. First the optimal values of C and gamma were estimated using grid-search and then the model was fit using the value of optimal lambda.

Post successful fitting of the model, the polarity (labels) on the test dataset was estimated using the model and was checked against the true polarity. Based on this, Accuracy, Precision-Score, Recall Score and F1-Score were calculated.

The metrics is tabulated below

Model	**Best C**	**Best gamma**	**Test Accuracy**	**Precision Score**	**Recall Score**	**F1 Score**
Bag-of-Words	100	0.1	86 %	0.90	0.95	0.92
TF-IDF	100	1	92 %	0.93	0.99	0.96
Average Word2Vec	10000	0.1	89 %	0.91	0.97	0.94
Weighted Word2Vec	10000	0.1	89 %	0.97	0.91	0.94

</body> </html>

```
In [ ]:
```