KnnModelForAmazonFoodReview (1)

April 7, 2020

KNN model implementation for amazon review dataset using BOW, tf-idf, Avg Word2Vec, tfidf- Word2Vec

[1]:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import sqlite3
from tqdm import tqdm
import  warnings
warnings.filterwarnings('ignore')
import string
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
import pickle
%matplotlib inline
```

[2]:

```
con = sqlite3.connect('/home/niranjan/Downloads/database.sqlite')
data = pd.read_sql_query('select * from Reviews where Score!=3 ',con)
data['Score'] = [1 if i > 3 else 0 for i in data['Score']]
#print(data.shape)
```

[3]:

```
data_sort = data.sort_values(by='ProductId',ascending=True, inplace=False,␣
 ↪kind='quicksort')
data_groupby = data_sort[['ProductId','Score']].groupby('Score').count()
```

```
#print(data_groupby)
```

Removing duplicates data

[4]:
```
data_without_dup = data_sort.
 ↪drop_duplicates(subset={'UserId','ProfileName','Time','Text'},keep='first',inplace=False)
#print(data_without_dup.shape)
data_groupby = data_without_dup[['ProductId','Score']].groupby('Score').count()
#print(data_groupby)
```

[5]:
```
data_without_dup = data_without_dup[data_without_dup.
 ↪HelpfulnessNumerator<=data_without_dup.HelpfulnessDenominator]
#print(data_without_dup.shape)
```

[6]:
```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
stop = set(stopwords.words('english'))
lemma = nltk.WordNetLemmatizer()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/niranjan/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/niranjan/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[7]:
```
import re
def cleanhtml(words):
  tag = re.compile(r'<.?>')
  cleanSent = re.sub(tag,'',words)
  return cleanSent

def PuncRemov(words):
  tag = re.compile(r'[^a-zA-Z]')
  cleanSent = re.sub(tag,'',words)
  return cleanSent
```

****————————-data cleaning————————-****

---

removal of html tags, symbols other than alphabets, stopwords and performing lemmatization as part of data pre-processing.

Lemmatization :- is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary

form.**

[8]:

```
final_string = []
str1 = ' '
positive_word = []
negative_word = []
i=0
for sen in data_without_dup['Text'].values:
  filtered_word = []
  sent = cleanhtml(sen)
  for word in sent.split():
    cleanWord = PuncRemov(word)
    for cleaned_words in cleanWord.split():
      if ((len(cleaned_words) > 2) & (cleaned_words.isalpha())):
        if (cleaned_words.islower() not in stop):
          w = (lemma.lemmatize(cleaned_words.lower())).encode('utf8')
          filtered_word.append(w)
          if data_without_dup['Score'].values[i]=='positive':
            positive_word.append(w)
          else:
            negative_word.append(w)
        else:
          continue
      else:
        continue
  str1 = b" ".join(filtered_word)
  final_string.append(str1)
  i = i+1
```

[9]:

```
data_without_dup['cleaned_text'] = final_string
data_without_dup['cleaned_text'] = data_without_dup['cleaned_text'].str.
 ↪decode('utf8')
```

sorting data based on time in ascending order

[10]:

```
data_without_dup = data_without_dup.
 ↪sort_values(by='Time',ascending=True,inplace=False)
```

[11]:

```
X = data_without_dup['cleaned_text']
y = data_without_dup['Score']
```

[12]:

```
X_train= X[0:100000]
y_train = y[0:100000]
X_cv= X[100000:120000]
y_cv = y[100000:120000]
X_test= X[120000:140000]
y_test = y[120000:140000]
```

[13]:

```
print("null_accuracy :",max(y.mean(),(1-y.mean())))
```

```
null_accuracy : 0.843178067446337
```

pre-processing of text using BOW

[21]:

```
count_vect = CountVectorizer()
X_train_bow = count_vect.fit_transform(X_train)
X_test_bow = count_vect.transform(X_test)
X_cv_bow = count_vect.transform(raw_documents=X_cv)
print(X_cv_bow.shape)
```

```
(20000, 88686)
```

[ ]:

```
import pickle
pickle_in1 = open("/home/niranjan/Downloads/AppliedAI/X_train_bow","rb")
pickle_in2 = open("/home/niranjan/Downloads/AppliedAI/X_cv_bow","rb")
pickle_in3 = open("/home/niranjan/Downloads/AppliedAI/X_test_bow","rb")

X_train_bow = pickle.load(pickle_in1)
X_cv_bow = pickle.load(pickle_in2)
X_test_bow = pickle.load(pickle_in3)
pickle_in1.close()
pickle_in2.close()
pickle_in3.close()
```

Best fit using Cross validation

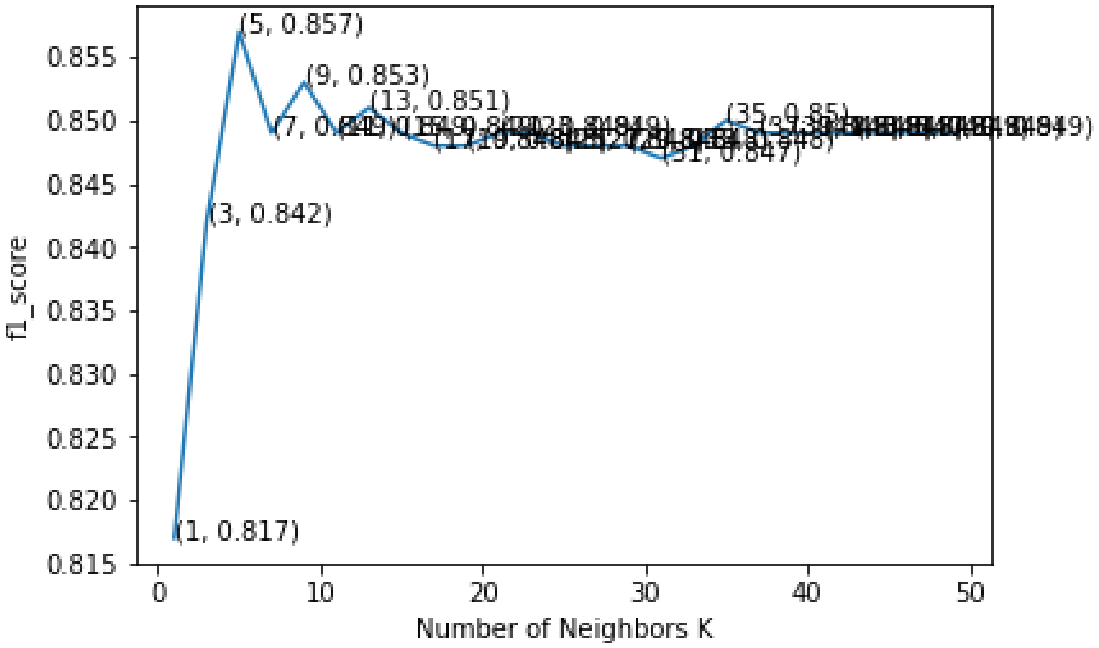
[74]:

```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]


cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',n_jobs=1)
    classifier.fit(X_train_bow,y_train)
    for i in range(0,X_cv_bow.shape[0],1000):
        y_pred = []
        y_pred.append(classifier.predict(X_cv_bow[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k = neighbor[cv_scores.index(max(cv_scores))]
print("optimal number of is: ", optimal_k)
```

```
plt.plot(neighbor,cv_scores)
for xy in zip(neighbor, np.round(cv_scores,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

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

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
```

```
optimal number of is:  5
```



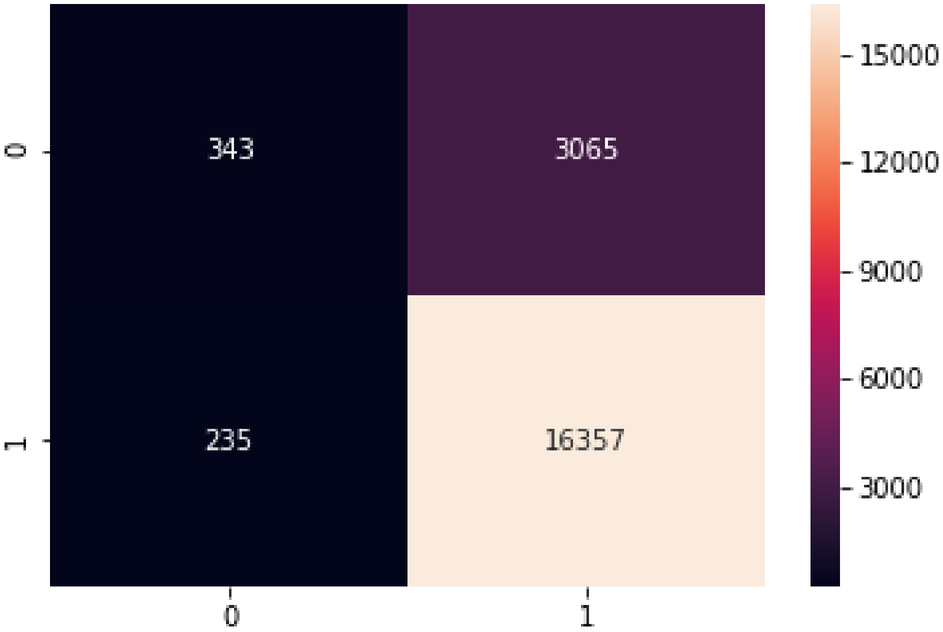
[153]:

```
y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k)
clf.fit(X_train_bow,y_train)
for i in range(0,X_test_bow.shape[0],1000):
    y_pred.append(clf.predict(X_test_bow[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
```

```
acc = f1_score(y_test, y_pred_flat,average='micro')*100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k,␣
 ↪acc))
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_flat, pos_label=2)
df = pd.DataFrame(confusion_matrix(y_test,y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 5 is 83.500000%
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```

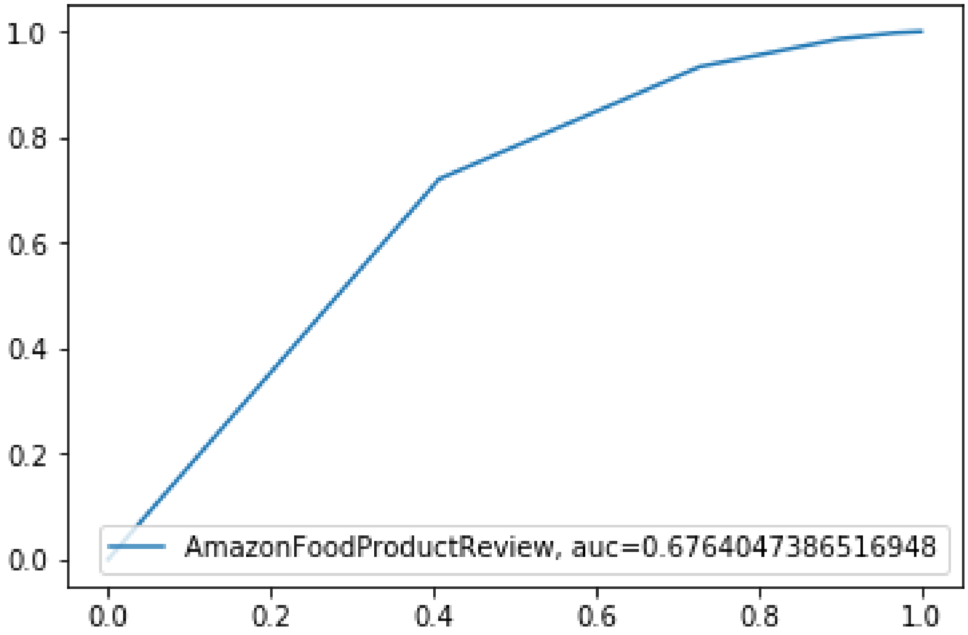


[144]:
```
y_pred_proba = []
for i in range(0,X_test_bow.shape[0],1000):
    y_pred_proba.append(clf.predict_proba(X_test_bow[i:i+1000]))
```

[145]:
```
y_pred = []
for i in y_pred_proba:
    for j in i:
        y_pred.append(j[1])
```

[146]:

```
fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

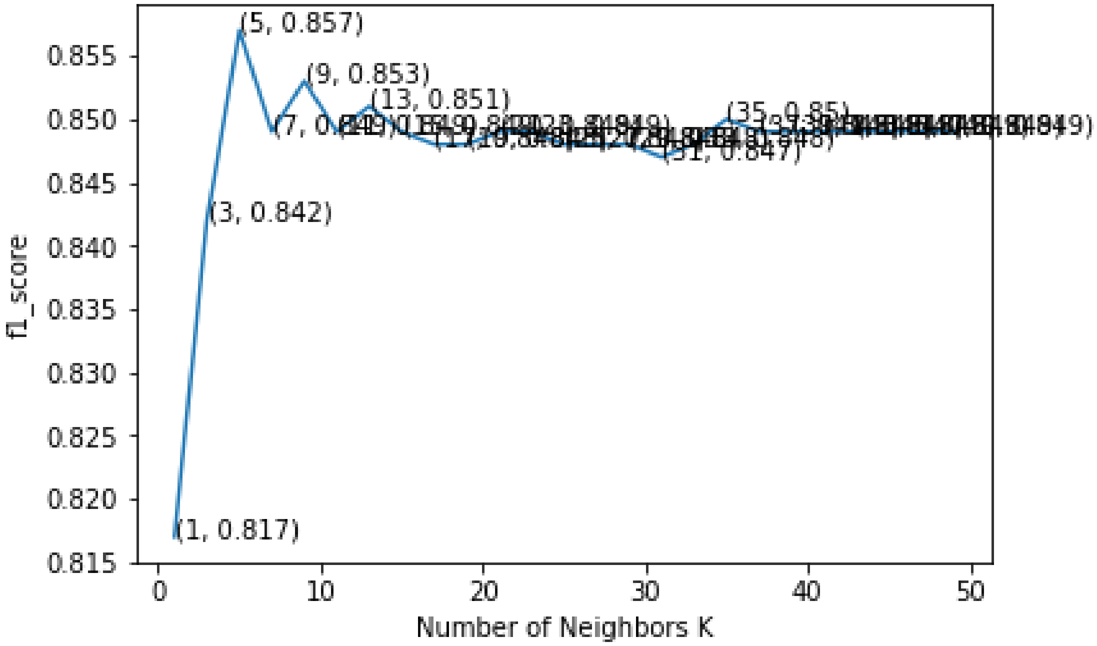


[147]:

```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]


cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='brute',n_jobs=1)
    classifier.fit(X_train_bow,y_train)
    for i in range(0,X_cv_bow.shape[0],1000):
        y_pred = []
        y_pred.append(classifier.predict(X_cv_bow[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k_brute = neighbor[cv_scores.index(max(cv_scores))]
```

```
print("optimal number of is: ", optimal_k_brute)

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

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
optimal number of is:  5
```

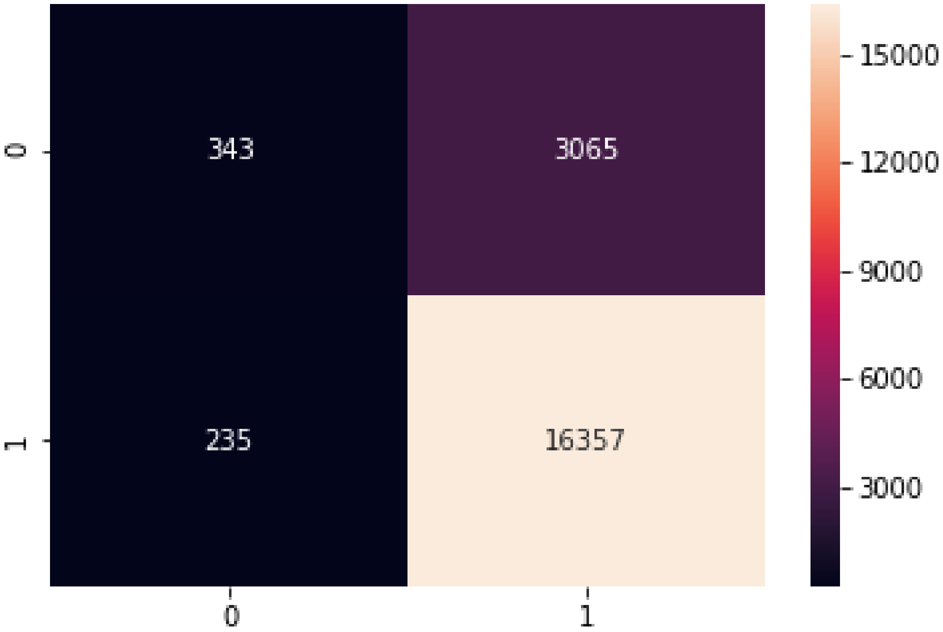


[152]:
```
y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k_brute)
clf.fit(X_train_bow,y_train)
for i in range(0,X_test_bow.shape[0],1000):
    y_pred.append(clf.predict(X_test_bow[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
acc = f1_score(y_test, y_pred_flat,average='micro')*100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
 ↪(optimal_k_brute, acc))
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_flat, pos_label=2)
```

```
df = pd.DataFrame(confusion_matrix(y_test,y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 5 is 83.500000%
```

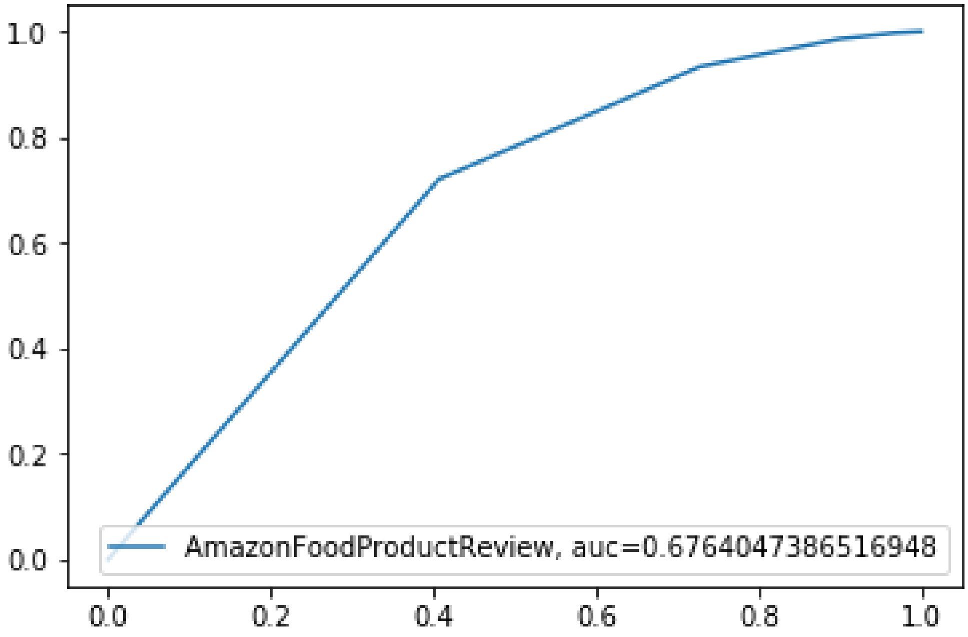
```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```



```
[149]: y_pred_proba = []
       for i in range(0,X_test_bow.shape[0],1000):
           y_pred_proba.append(clf.predict_proba(X_test_bow[i:i+1000]))
```

```
[150]: y_pred = []
       for i in y_pred_proba:
           for j in i:
               y_pred.append(j[1])
```

```
[151]: fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred)
       auc = metrics.roc_auc_score(y_test, y_pred)
       plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
       plt.legend(loc=4)
       plt.show()
```



Preprocessing using tf-idf

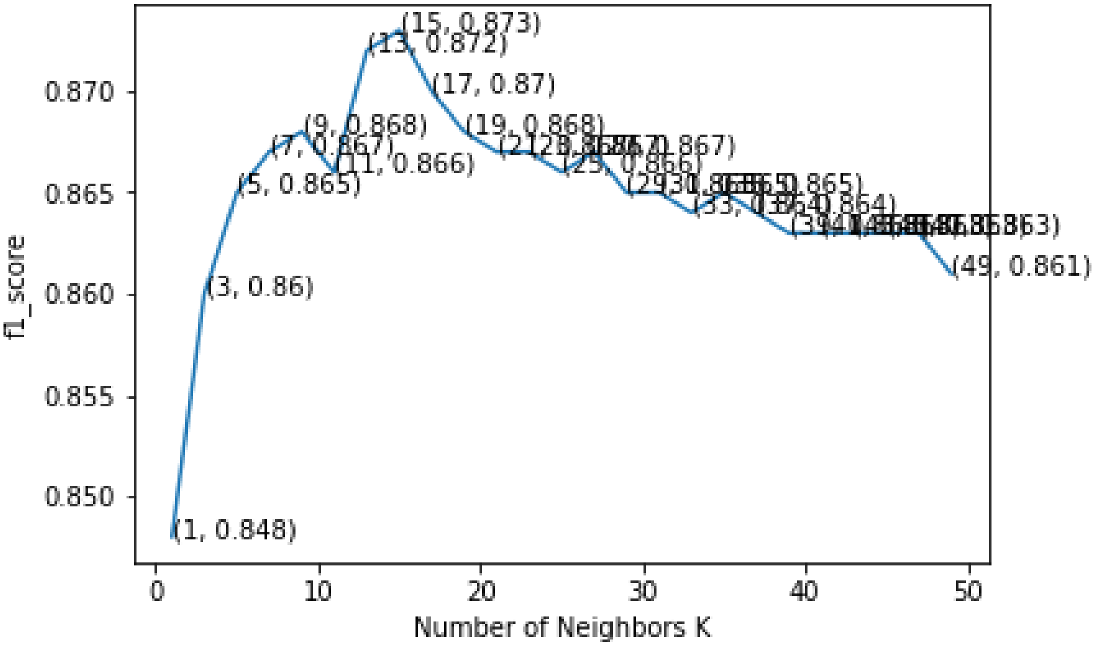
[154]:
```
vec = TfidfVectorizer()
```

[157]:
```
X_train_tfidf = vec.fit_transform(X_train,y_train)
X_cv_tfidf = vec.transform(X_cv)
X_test_tfidf = vec.transform(X_test)
```

[163]:
```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]


cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='brute',n_jobs=1)
    classifier.fit(X_train_tfidf,y_train)
    for i in range(0,X_cv_tfidf.shape[0],1000):
        y_pred = []
        y_pred.append(classifier.predict(X_cv_tfidf[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k
```

```
optimal_k_brute = neighbor[cv_scores.index(max(cv_scores))]
print("optimal number of is: ", optimal_k_brute)

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

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
optimal number of is:  15
```

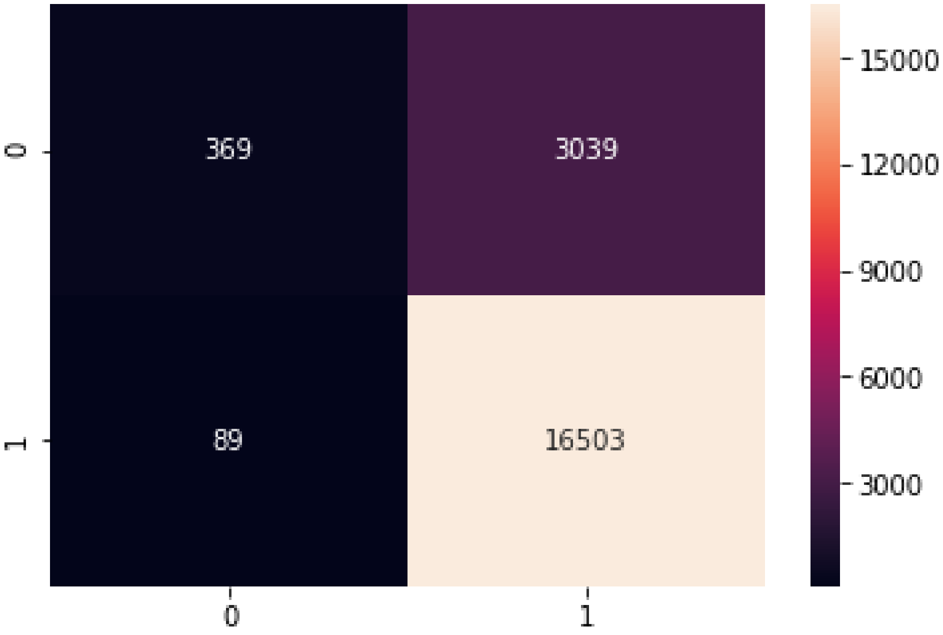


[164]:
```
y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k_brute)
clf.fit(X_train_tfidf,y_train)
for i in range(0,X_test_tfidf.shape[0],1000):
    y_pred.append(clf.predict(X_test_tfidf[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
acc = f1_score(y_test, y_pred_flat,average='micro')*100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
 ↪(optimal_k_brute, acc))
```

```
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_flat, pos_label=2)
df = pd.DataFrame(confusion_matrix(y_test,y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 15 is 84.360000%
```

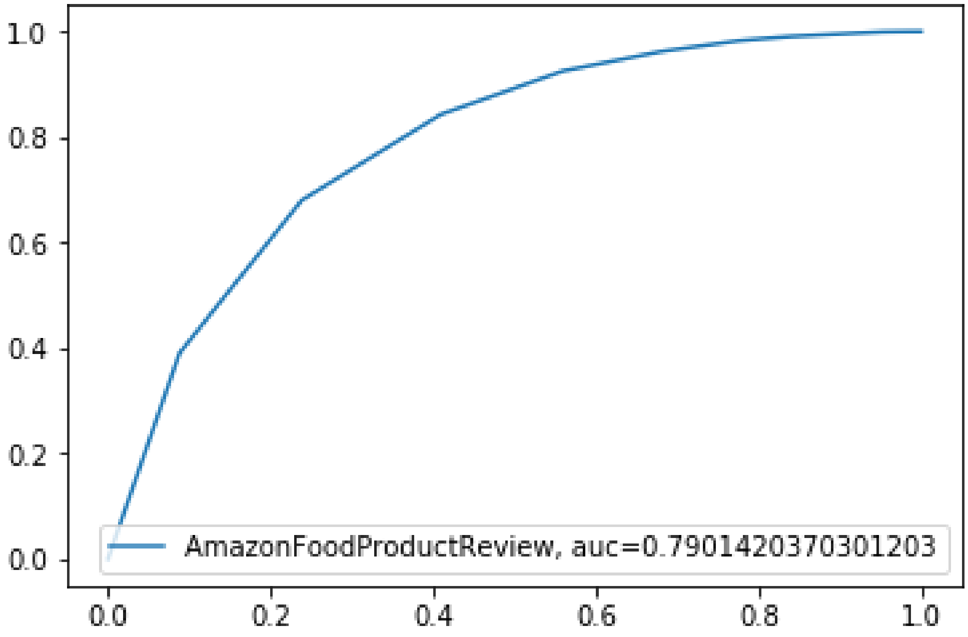
```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```



[165]:
```
y_pred_proba = []
for i in range(0,X_test_tfidf.shape[0],1000):
    y_pred_proba.append(clf.predict_proba(X_test_tfidf[i:i+1000]))
```

[166]:
```
y_pred = []
for i in y_pred_proba:
    for j in i:
        y_pred.append(j[1])
```

[167]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)
plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Algorithm as kd_tree

[168]:
```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]


cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',n_jobs=1)
    classifier.fit(X_train_tfidf,y_train)
    for i in range(0,X_cv_tfidf.shape[0],1000):
        y_pred = []
        y_pred.append(classifier.predict(X_cv_tfidf[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k_kd_tree = neighbor[cv_scores.index(max(cv_scores))]
print("optimal number of k is: ", optimal_k_kd_tree)

plt.plot(neighbor,cv_scores)
```

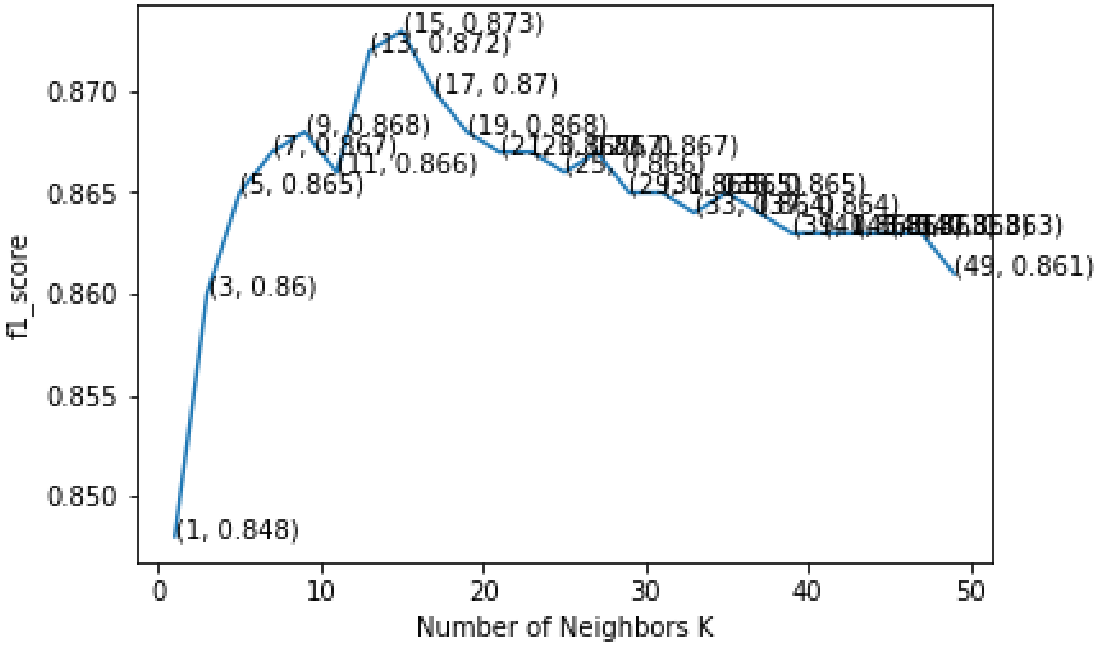
```
for xy in zip(neighbor, np.round(cv_scores,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
```

```
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
```

```
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/neighbors/base.py:212: UserWarning: cannot use tree with sparse
input: using brute force
  warnings.warn("cannot use tree with sparse input: "
```

```
optimal number of is:  15
```

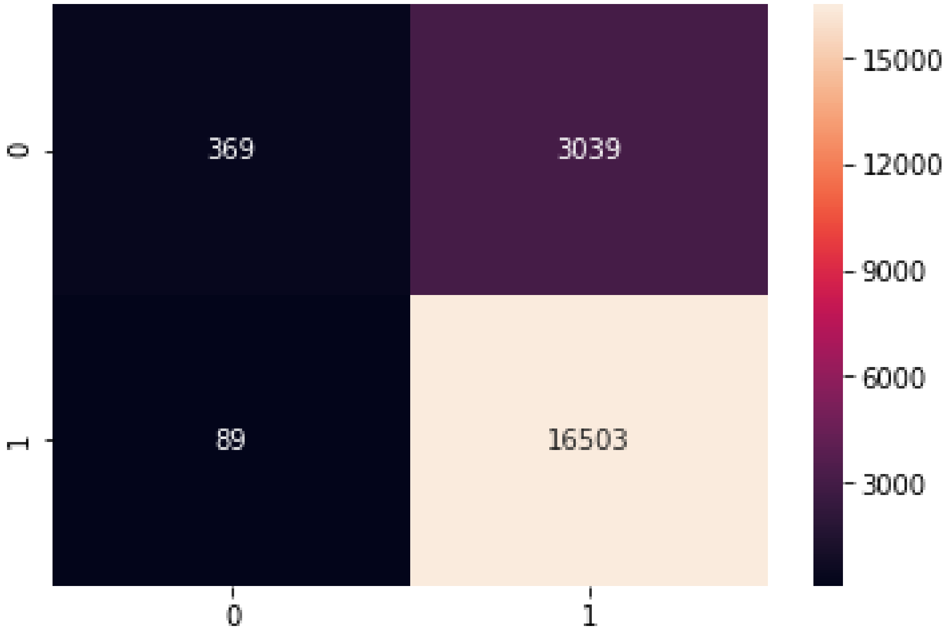


```
[169]: y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k_kd_tree)
clf.fit(X_train_tfidf,y_train)
for i in range(0,X_test_tfidf.shape[0],1000):
    y_pred.append(clf.predict(X_test_tfidf[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
acc = f1_score(y_test, y_pred_flat,average='micro')*100
```

```
print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
↪(optimal_k_kd_tree, acc))
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_flat, pos_label=2)
df = pd.DataFrame(confusion_matrix(y_test,y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 15 is 84.360000%
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```

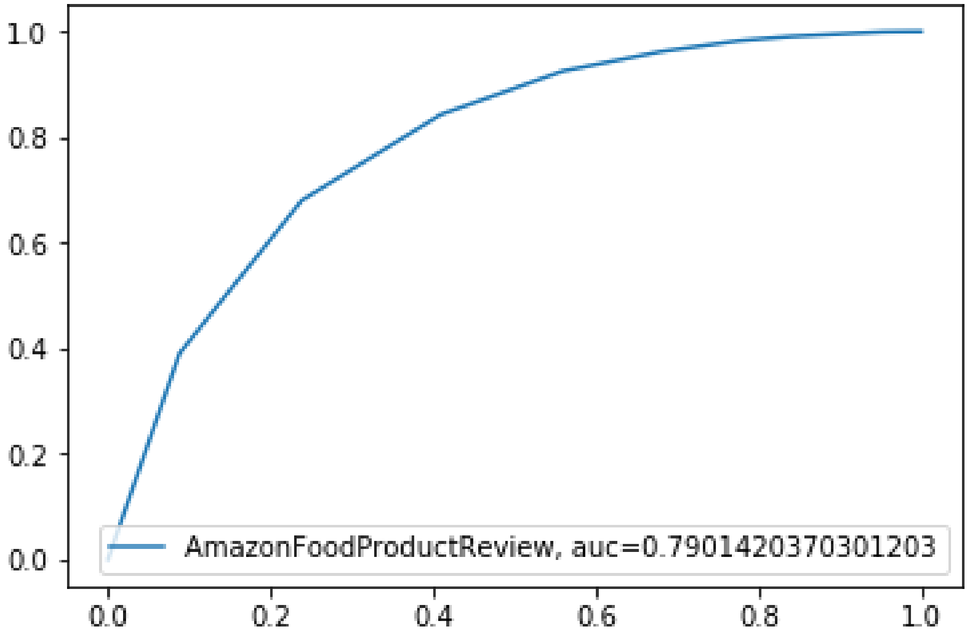


```
[170]: y_pred_proba = []
       for i in range(0,X_test_tfidf.shape[0],1000):
           y_pred_proba.append(clf.predict_proba(X_test_tfidf[i:i+1000]))
```

```
[171]: y_pred = []
       for i in y_pred_proba:
           for j in i:
               y_pred.append(j[1])
```

```
[172]: fpr, tpr, thresholds = metrics.roc_curve(y_test,  y_pred)
       auc = metrics.roc_auc_score(y_test, y_pred)
```

```
plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



[14]:

```
list_of_sent = []

for sent in data_without_dup['cleaned_text'].values:
  list_of_sent.append(sent.split())
```

[15]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

[16]:

```
model = KeyedVectors.load_word2vec_format('/home/niranjan/Downloads/data',␣
 ↪binary=True)
```

[33]:

```
X_train_avg_w2v = []
w2v_model = Word2Vec(list_of_sent[0:100000],min_count=5,size=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)

for sent in list_of_sent[0:100000]:
    sent_vec = np.zeros(100)
    count_words = 0
    for words in sent:
        if words in  w2v_words:
            vec = w2v_model.wv[words]
            sent_vec += vec
            count_words +=1
    if count_words !=0:
        sent_vec = sent_vec/count_words
    X_train_avg_w2v.append(sent_vec)
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/gensim/models/base_any2vec.py:743: UserWarning: C extension not loaded,
training will be slow. Install a C compiler and reinstall gensim for fast
training.
  "C extension not loaded, training will be slow. "
```

[34]:

```
X_cv_avg_w2v = []
w2v_model = Word2Vec(list_of_sent[100000:120000],min_count=5,size␣
 ↪=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in list_of_sent[100000:120000]:
    sent_vec = np.zeros(100)
    count_words = 0
    for words in sent:
        if words in  w2v_words:
            vec = w2v_model.wv[words]
            sent_vec += vec
            count_words +=1
    if count_words !=0:
        sent_vec = sent_vec/count_words
    X_cv_avg_w2v.append(sent_vec)
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/gensim/models/base_any2vec.py:743: UserWarning: C extension not loaded,
training will be slow. Install a C compiler and reinstall gensim for fast
training.
  "C extension not loaded, training will be slow. "
```

[35]:

```
X_test_avg_w2v = []
w2v_model = Word2Vec(list_of_sent[120000:140000],min_count=5,size␣
 ↪=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in list_of_sent[120000:140000]:
    sent_vec = np.zeros(100)
    count_words = 0
    for words in sent:
        if words in  w2v_words:
            vec = w2v_model.wv[words]
            sent_vec += vec
            count_words +=1
    if count_words !=0:
        sent_vec = sent_vec/count_words
    X_test_avg_w2v.append(sent_vec)
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/gensim/models/base_any2vec.py:743: UserWarning: C extension not loaded,
training will be slow. Install a C compiler and reinstall gensim for fast
training.
  "C extension not loaded, training will be slow. "
```

[36]:

```
import pickle
pickle_in1 = open("/home/niranjan/Downloads/AppliedAI/X_train_avg_w2v","wb")
pickle_in2 = open("/home/niranjan/Downloads/AppliedAI/X_cv_avg_w2v","wb")
pickle_in3 = open("/home/niranjan/Downloads/AppliedAI/X_test_avg_w2v","wb")

pickle.dump(X_train_avg_w2v,pickle_in1)
pickle.dump(X_cv_avg_w2v,pickle_in2)
pickle.dump(X_test_avg_w2v,pickle_in3)
pickle_in1.close()
pickle_in2.close()
pickle_in3.close()
```

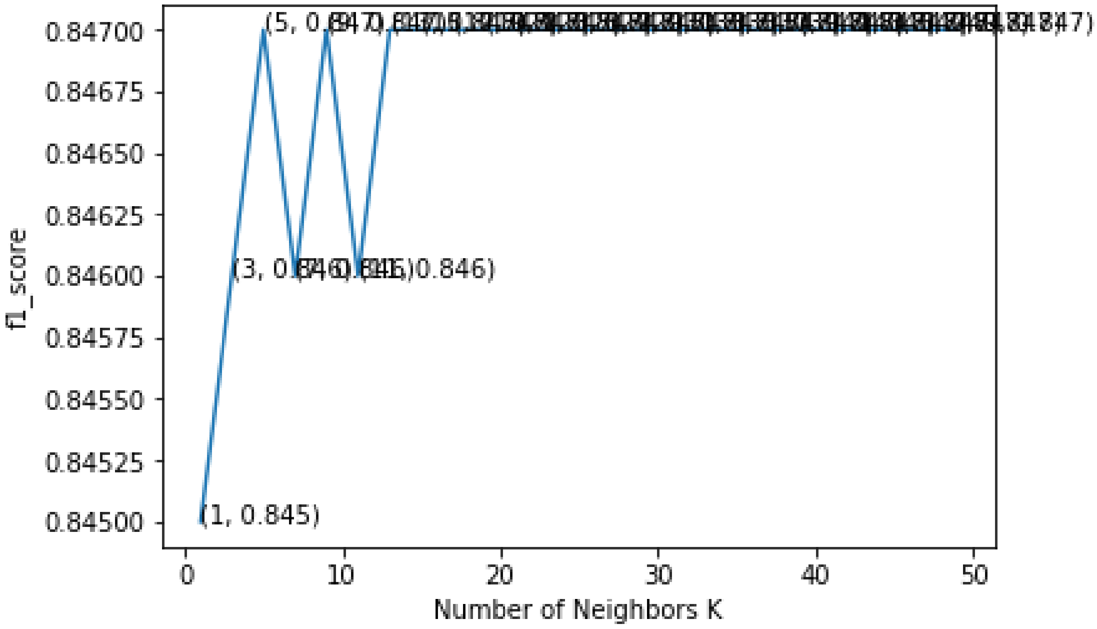
[1]:

```
import pickle
pickle_out1 = open("/home/niranjan/Downloads/AppliedAI/X_train_avg_w2v","rb")
pickle_out2 = open("/home/niranjan/Downloads/AppliedAI/X_cv_avg_w2v","rb")
pickle_out3 = open("/home/niranjan/Downloads/AppliedAI/X_test_avg_w2v","rb")

X_train_avg_w2v = pickle.load(pickle_out1)
X_cv_avg_w2v = pickle.load(pickle_out2)
X_test_avg_w2v = pickle.load(pickle_out3)
pickle_out1.close()
pickle_out2.close()
pickle_out3.close()
```

[15]:

```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]


cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='brute',n_jobs=1)
    classifier.fit(X_train_avg_w2v,y_train[0:100000])
    for i in range(0,len(X_cv_avg_w2v),1000):
        y_pred = []
        y_pred.append(classifier.predict(X_cv_avg_w2v[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k_brute = neighbor[cv_scores.index(max(cv_scores))]
print("optimal number of k is: ", optimal_k_brute)

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

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
optimal number of k is:  5
```

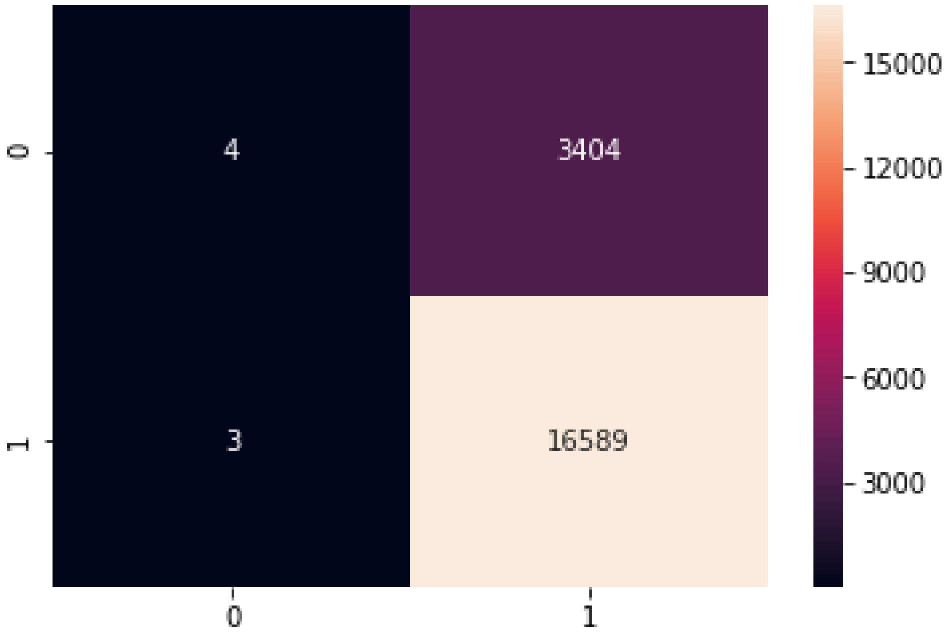


[38]:
```
y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k_brute)
clf.fit(X_train_avg_w2v,y_train[0:100000])
for i in range(0,len(X_test_avg_w2v),1000):
    y_pred.append(clf.predict(X_test_avg_w2v[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
```

```
acc = f1_score(y_test[0:20000], y_pred_flat,average='micro')*100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
↪(optimal_k_brute, acc))
fpr, tpr, thresholds = metrics.roc_curve(y_test[0:20000], y_pred_flat,␣
↪pos_label=2)
df = pd.DataFrame(confusion_matrix(y_test[0:20000],y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 5 is 82.965000%
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```

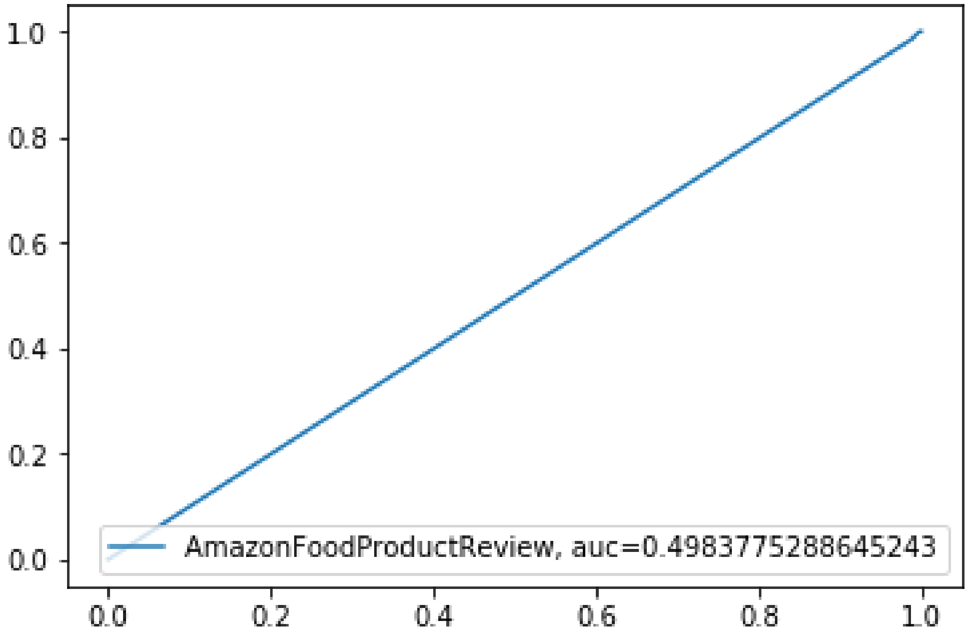


[41]:
```
y_pred_proba = []
for i in range(0,len(X_test_avg_w2v),1000):
    y_pred_proba.append(clf.predict_proba(X_test_avg_w2v[i:i+1000]))
```

[42]:
```
y_pred = []
for i in y_pred_proba:
    for j in i:
        y_pred.append(j[1])
```

[43]:

```
fpr, tpr, thresholds = metrics.roc_curve(y_test[0:20000],  y_pred)
auc = metrics.roc_auc_score(y_test[0:20000], y_pred)
plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

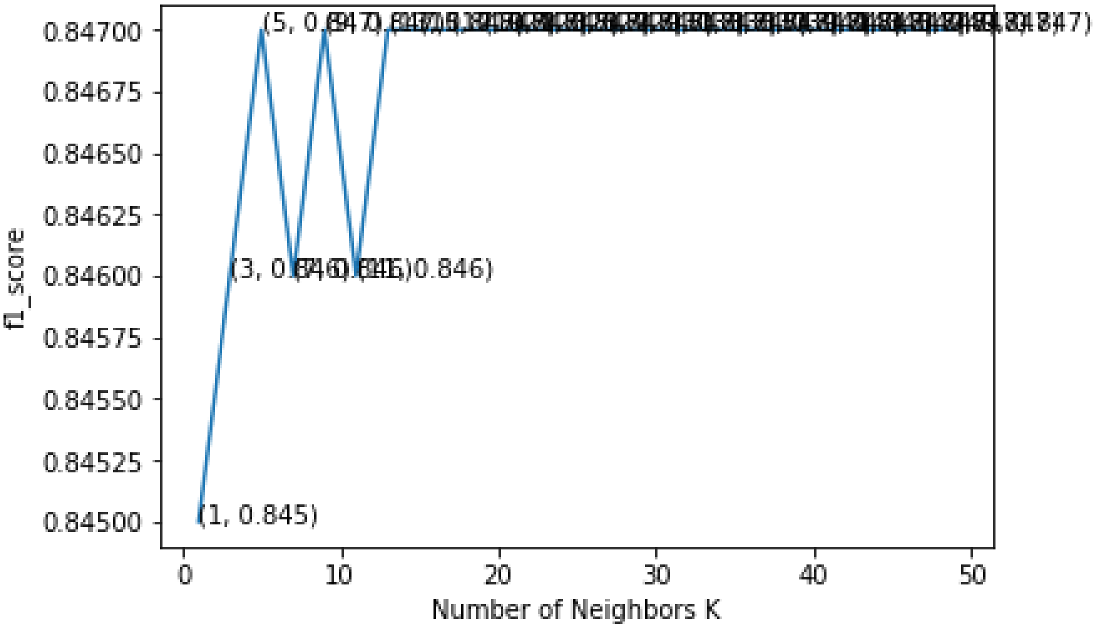


[44]:

```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]


cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',n_jobs=1)
    classifier.fit(X_train_avg_w2v,y_train[0:100000])
    for i in range(0,len(X_cv_avg_w2v),1000):
        y_pred = []
        y_pred.append(classifier.predict(X_cv_avg_w2v[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k_kd_tree = neighbor[cv_scores.index(max(cv_scores))]
```

```
print("optimal number of k is: ", optimal_k_kd_tree)

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

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
optimal number of k is:  5
```



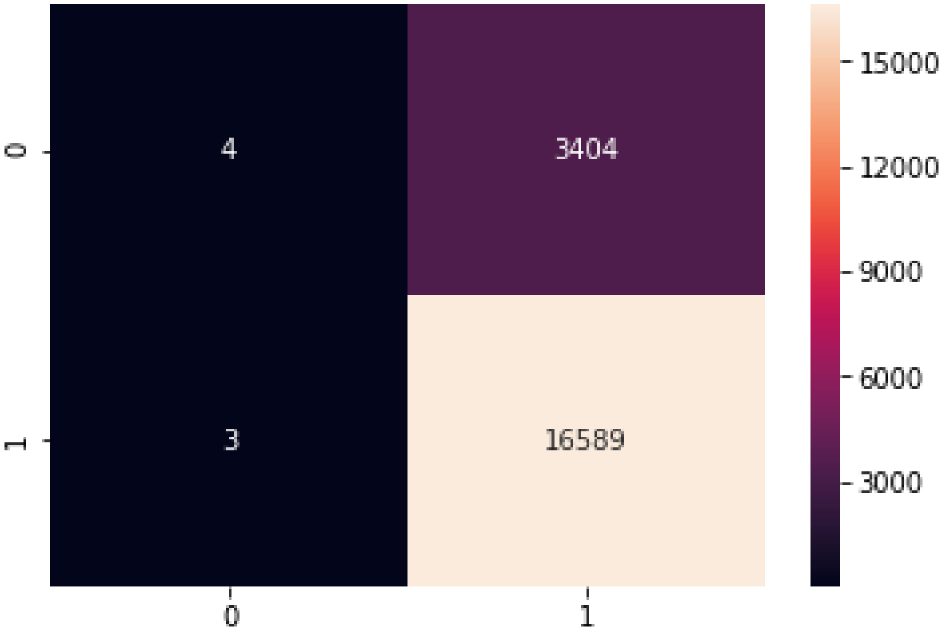
[45]:

```
y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k_kd_tree)
clf.fit(X_train_avg_w2v,y_train[0:100000])
for i in range(0,len(X_test_avg_w2v),1000):
    y_pred.append(clf.predict(X_test_avg_w2v[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
acc = f1_score(y_test[0:20000], y_pred_flat,average='micro')*100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
 ↪(optimal_k_kd_tree, acc))
fpr, tpr, thresholds = metrics.roc_curve(y_test[0:20000], y_pred_flat,␣
 ↪pos_label=2)
```

```
df = pd.DataFrame(confusion_matrix(y_test[0:20000],y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 5 is 82.965000%
```

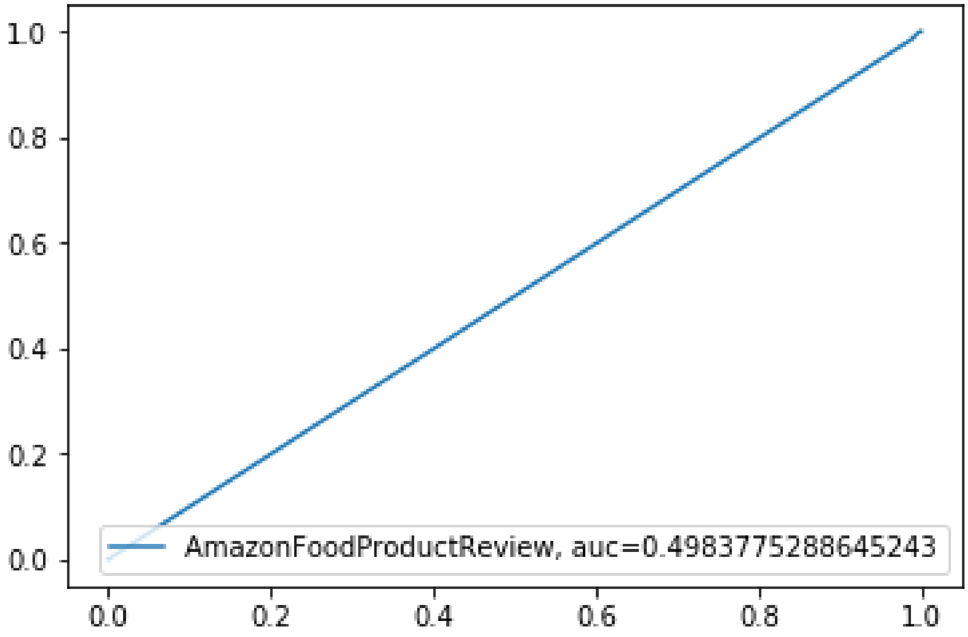
```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```



```
[46]: y_pred_proba = []
      for i in range(0,len(X_test_avg_w2v),1000):
          y_pred_proba.append(clf.predict_proba(X_test_avg_w2v[i:i+1000]))
```

```
[47]: y_pred = []
      for i in y_pred_proba:
          for j in i:
              y_pred.append(j[1])
```

```
[48]: fpr, tpr, thresholds = metrics.roc_curve(y_test[0:20000],  y_pred)
      auc = metrics.roc_auc_score(y_test[0:20000], y_pred)
      plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
      plt.legend(loc=4)
      plt.show()
```



tfidf-Word2vec as vectorizer

[17]:
```
# TF-IDF weighted Word2Vec
vec = TfidfVectorizer()
vec.fit_transform(X_train,y_train)
tfidf_feat = vec.get_feature_names() # tfidf words/col-names
dictionary = dict(zip(vec.get_feature_names(), list(vec.idf_)))
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val =␣
↪tfidf
tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is stored in␣
↪this list
row=0;
w2v_model = Word2Vec(list_of_sent[0:100000],min_count=5,size=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in tqdm(list_of_sent[0:100000]): # for each review/sentence
    sent_vec = np.zeros(100) # 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:
            if word in tfidf_feat:
                vect = 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 += (vect * tf_idf)
                weight_sum += tf_idf
            else:
                break
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_train_vectors.append(sent_vec)
    row += 1
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/gensim/models/base_any2vec.py:743: UserWarning: C extension not loaded,
training will be slow. Install a C compiler and reinstall gensim for fast
training.
  "C extension not loaded, training will be slow. "
100%|      | 100000/100000 [3:21:42<00:00,  6.19it/s]
```

[18]:

```
# TF-IDF weighted Word2Vec
vec = TfidfVectorizer()
vec.fit_transform(X_train,y_train)

tfidf_feat = vec.get_feature_names() # tfidf words/col-names
dictionary = dict(zip(vec.get_feature_names(), list(vec.idf_)))
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val =␣
 ↪tfidf
tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stored in␣
 ↪this list
row=0;
w2v_model = Word2Vec(list_of_sent[100000:120000],min_count=5,size=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in tqdm(list_of_sent[100000:120000]): # for each review/sentence
    sent_vec = np.zeros(100) # 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:
            if 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
            else:
                break
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_cv_vectors.append(sent_vec)
    row += 1
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/gensim/models/base_any2vec.py:743: UserWarning: C extension not loaded,
training will be slow. Install a C compiler and reinstall gensim for fast
training.
  "C extension not loaded, training will be slow. "
100%|      | 20000/20000 [39:48<00:00,  7.59it/s]
```

[19]:

```
# TF-IDF weighted Word2Vec
vec = TfidfVectorizer()
vec.fit_transform(X_train,y_train)

tfidf_feat = vec.get_feature_names() # tfidf words/col-names
dictionary = dict(zip(vec.get_feature_names(), list(vec.idf_)))
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val =␣
 ↪tfidf
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is stored in␣
 ↪this list
row=0;
w2v_model = Word2Vec(list_of_sent[120000:140000],min_count=5,size=100,workers=4)
w2v_words = list(w2v_model.wv.vocab)
for sent in tqdm(list_of_sent[120000:140000]): # for each review/sentence
    sent_vec = np.zeros(100) # 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:
            if 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
            else:
                break
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_test_vectors.append(sent_vec)
    row += 1
```

```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/gensim/models/base_any2vec.py:743: UserWarning: C extension not loaded,
training will be slow. Install a C compiler and reinstall gensim for fast
```

```
training.
  "C extension not loaded, training will be slow. "
100%|      | 20000/20000 [42:57<00:00, 10.68it/s]
```

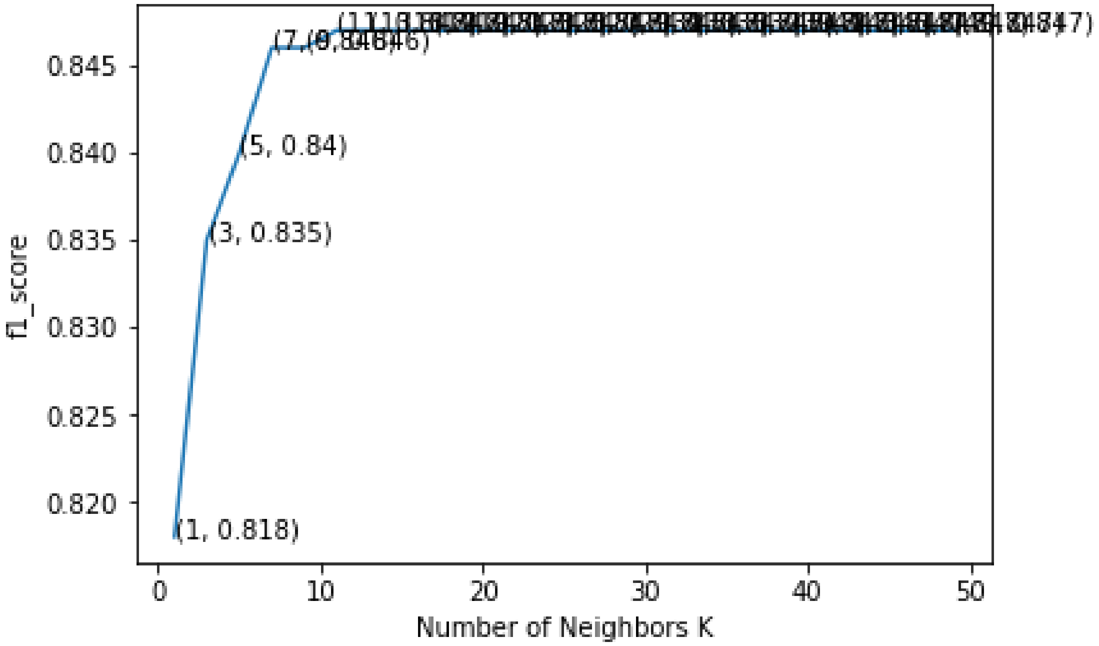
[21]:

```python
  import pickle
pickle_in1 = open("/home/niranjan/Downloads/AppliedAI/tfidf_train_vectors","wb")
pickle_in2 = open("/home/niranjan/Downloads/AppliedAI/tfidf_cv_vectors","wb")
pickle_in3 = open("/home/niranjan/Downloads/AppliedAI/tfidf_test_vectors","wb")

pickle.dump(tfidf_train_vectors,pickle_in1)
pickle.dump(tfidf_cv_vectors,pickle_in2)
pickle.dump(tfidf_test_vectors,pickle_in3)
pickle_in1.close()
pickle_in2.close()
pickle_in3.close()
```

[29]:

```python
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]



cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='brute',n_jobs=1)
    classifier.fit(tfidf_train_vectors,y_train[0:100000])
    for i in range(0,len(tfidf_cv_vectors),1000):
        y_pred = []
        y_pred.append(classifier.predict(tfidf_cv_vectors[i:i+1000]))
        y_pred_flat = [item for sublist in y_pred for item in sublist]
    cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k_tfidf_brute = neighbor[cv_scores.index(max(cv_scores))]
print("optimal number of k is: ", optimal_k_tfidf_brute)

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

plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
optimal number of k is:  11
```

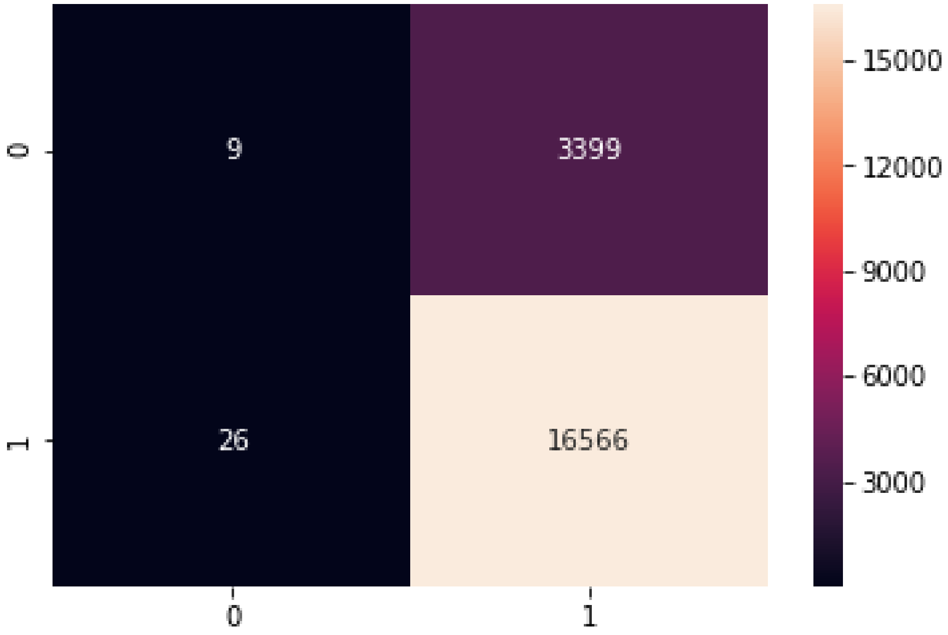


[33]:

```
y_pred = []
clf = KNeighborsClassifier(n_neighbors=optimal_k_tfidf_brute)
clf.fit(tfidf_train_vectors,y_train[0:200000])
for i in range(0,len(tfidf_test_vectors),1000):
    y_pred.append(clf.predict(tfidf_test_vectors[i:i+1000]))
# evaluate accuracy
y_pred_flat = [item for sublist in y_pred for item in sublist]
acc = f1_score(y_test[0:20000], y_pred_flat,average='micro')*100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
 ↪(optimal_k_tfidf_brute, acc))
fpr, tpr, thresholds = metrics.roc_curve(y_test[0:20000], y_pred_flat,␣
 ↪pos_label=2)
df = pd.DataFrame(confusion_matrix(y_test[0:20000],y_pred_flat))
sns.heatmap(df,annot=True,fmt="d")
plt.show()
```

```
The accuracy of the knn classifier for k = 11 is 82.875000%
```

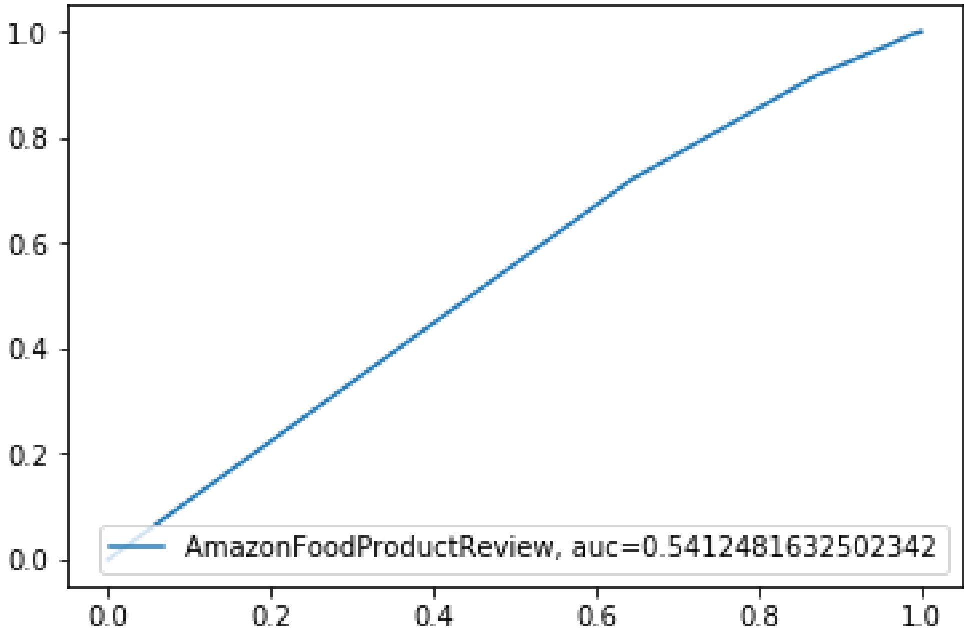
```
/home/niranjan/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/ranking.py:571: UndefinedMetricWarning: No positive
samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
```



[37]:
```
y_pred_proba = []
for i in range(0,len(tfidf_test_vectors),1000):
    y_pred_proba.append(clf.predict_proba(tfidf_test_vectors[i:i+1000]))
```

[38]:
```
y_pred = []
for i in y_pred_proba:
    for j in i:
        y_pred.append(j[1])
```

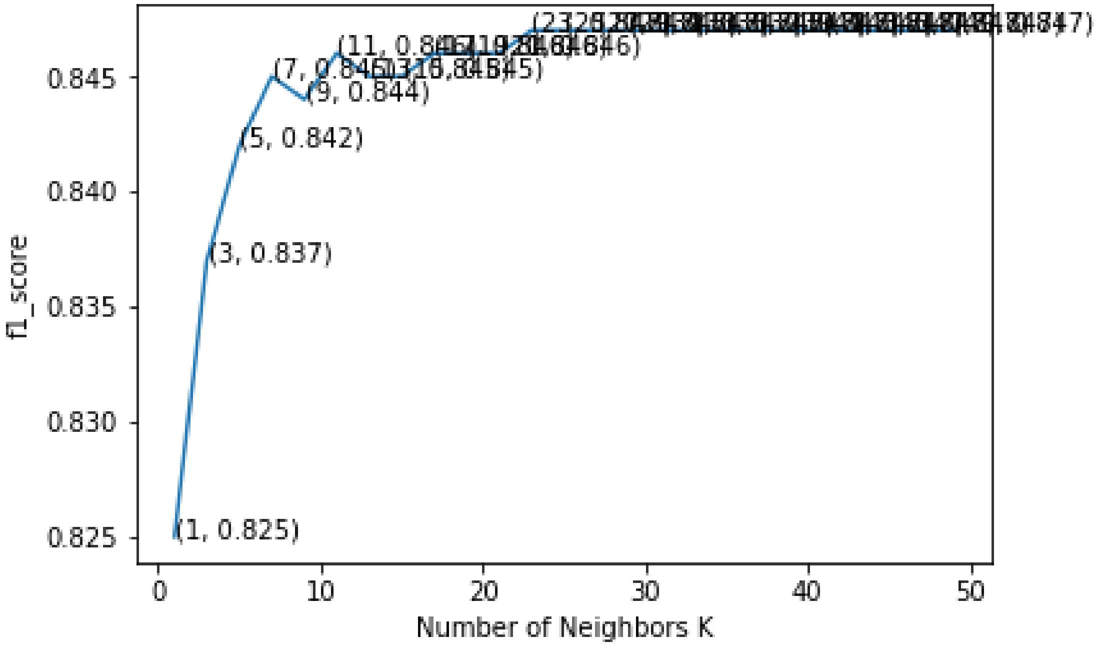
[39]:
```
fpr, tpr, thresholds = metrics.roc_curve(y_test[0:20000],  y_pred)
auc = metrics.roc_auc_score(y_test[0:20000], y_pred)
plt.plot(fpr,tpr,label="AmazonFoodProductReview, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



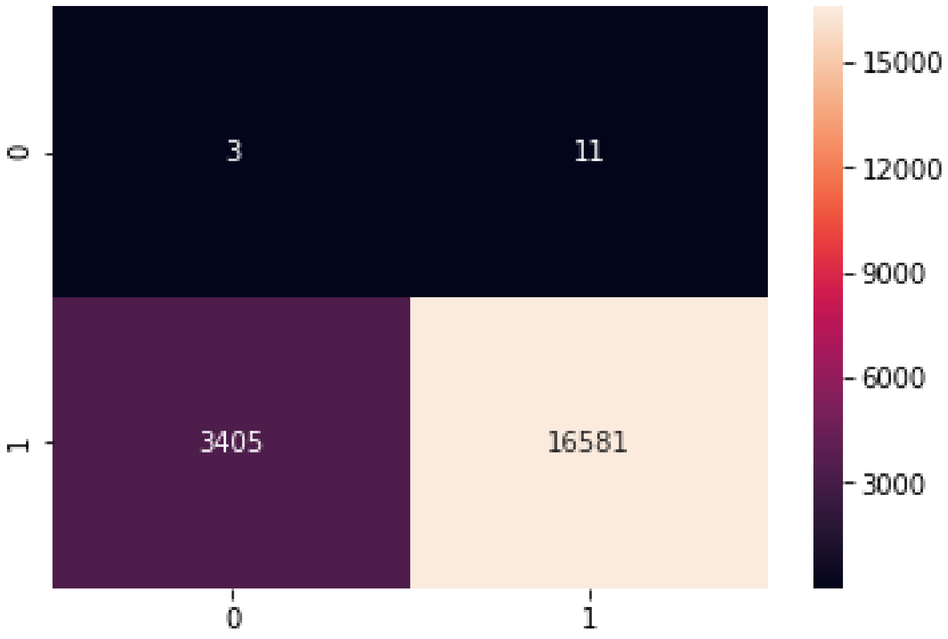
tfidf-Word2vec vectorizer with kd__tree as algorithm

[36]:
```
myList = list(range(0,50))

#creating odd list for k in Knn
neighbor = [x for x in myList if x%2 != 0]



cv_scores = []
for k in neighbor:
    classifier = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree',n_jobs=1)
     classifier.fit(tfidf_train_vectors,y_train[0:100000])
     for i in range(0,len(tfidf_test_vectors),1000):
         y_pred = []
         y_pred.append(classifier.predict(tfidf_test_vectors[i:i+1000]))
         y_pred_flat = [item for sublist in y_pred for item in sublist]
     cv_scores.append(f1_score(y_cv[i:i+1000],y_pred_flat,average='micro'))

#Determining optimal value of k

optimal_k_tfidf_kd_tree = neighbor[cv_scores.index(max(cv_scores))]
print("optimal number of k is: ", optimal_k_tfidf_kd_tree)

plt.plot(neighbor,cv_scores)
for xy in zip(neighbor, np.round(cv_scores,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel('Number of Neighbors K')
plt.ylabel('f1_score')
plt.show()

# print("f1_score for each k value is : ", np.round(cv_scores,3))
```

```
optimal number of k is:  23
```



```
[40]: clf = KNeighborsClassifier(n_neighbors=optimal_k_tfidf_kd_tree)
      clf.fit(tfidf_train_vectors,y_train[0:200000])
      pred = clf.predict(tfidf_test_vectors)
      # evaluate accuracy
      acc = f1_score(pred,y_test[0:20000])*100
      df = pd.DataFrame(confusion_matrix(pred,y_test[0:20000]))
      sns.heatmap(df,annot=True,fmt="d")
      plt.show()
      print('\nThe accuracy of the knn classifier for k = %d is %f%%' %␣
       ↪(optimal_k_tfidf_kd_tree, acc))
```



```
The accuracy of the knn classifier for k = 23 is 0.906611%
```

[2]:

```python
from prettytable import PrettyTable
x= PrettyTable()
x.field_names = ["Vectorizer","Model","Hyper parameter","Acurracy","Auc"]
x.add_row(["BOW","Brute","9","93.54",".6764"])
x.add_row(["BOW","kd_tree","9","93.54",".6764"])
x.add_row(["tf-idf","Brute","7","93.89",".79014"])
x.add_row(["tf-idf","kd_tree","7","93.89",".79014"])
x.add_row(["Avg-Word2Vec","Brute","13","94.21",".4983"])
x.add_row(["Avg-Word2Vec","kd_tree","5","82.9650",".4983"])
x.add_row(["tfidf-Word2Vec","Brute","11","82.8750",".5412"])
x.add_row(["tfidf-Word2Vec","kd_tree","23","90.66",".5412"])
print(x)
```

```
+----------------+---------+-----------------+----------+--------+
|   Vectorizer   |  Model  | Hyper parameter | Acurracy |  Auc   |
+----------------+---------+-----------------+----------+--------+
|      BOW       |  Brute  |        9        |  93.54   | .6764  |
|      BOW       | kd_tree |        9        |  93.54   | .6764  |
|     tf-idf     |  Brute  |        7        |  93.89   | .79014 |
|     tf-idf     | kd_tree |        7        |  93.89   | .79014 |
|  Avg-Word2Vec  |  Brute  |        13       |  94.21   | .4983  |
|  Avg-Word2Vec  | kd_tree |        5        | 82.9650  | .4983  |
| tfidf-Word2Vec |  Brute  |        11       | 82.8750  | .5412  |
| tfidf-Word2Vec | kd_tree |        23       |  90.66   | .5412  |
+----------------+---------+-----------------+----------+--------+
```