Amazon_Fine_Food_Assignment

October 9, 2018

1 Amazon Fine Food Reviews Analysis

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

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

1.1 We will analyse only 10k reviews .

```
In [1]: %matplotlib inline

        import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
```

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

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

# Loading data from sql file
con = sqlite3.connect('database.sqlite')

filtered_data = pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score !=3""",con)

filtered_data.shape
```

Out[1]: (525814, 10)

In [2]:
```
filtered_data.head(5)
```

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

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
3         Cough Medicine  If you are looking for the secret ingredient i...
4            Great taffy  Great taffy at a great price.  There was a wid...
```

In [3]:
```
# Replacing score > 3 with positive and score < 3 with negative.
filtered_score = filtered_data['Score']
modified_score = filtered_score.map(lambda x:'positive' if x > 3 else 'negative')
filtered_data['Score'] = modified_score
filtered_data.head(5)
```

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

           HelpfulnessNumerator  HelpfulnessDenominator     Score        Time  \
        0                     1                       1  positive  1303862400
        1                     0                       0  negative  1346976000
        2                     1                       1  positive  1219017600
        3                     3                       3  negative  1307923200
        4                     0                       0  positive  1350777600

                         Summary                                               Text
        0  Good Quality Dog Food  I have bought several of the Vitality canned d...
        1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
        2  "Delight" says it all  This is a confection that has been around a fe...
        3         Cough Medicine  If you are looking for the secret ingredient i...
        4            Great taffy  Great taffy at a great price.  There was a wid...
```

2 EDA and Deduplication

```
In [4]: # In this type of datasets some duplicate reviews are also present which are given by
        # Let's sort this dataset on productid and then remove duplicate reviews.
        sorted_data = filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=Fals

        # Now we will remove duplicate reviews
        final_data = sorted_data.drop_duplicates(subset={'UserId','ProfileName','Time','Text'}
        final_data.shape

Out[4]: (364173, 10)

In [5]: # Sometime the HelpfulnessNumerator is > HelpfullnessDenominator. We will remove such
        final_data = final_data.loc[final_data['HelpfulnessNumerator']<=final_data['Helpfulnes
        final_data.shape

Out[5]: (364171, 10)
```

3 Text Preprocessing: Stemming, stop-word removal and Lemmatization.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags

2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [6]:
```
# Function to remove any html tags present in the review
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

stop_words = list(stopwords.words('english'))
stop = [x.replace('not',' ').replace('nor',' ') for x in stop_words]

# Initializing snowball stemmer
snow_ball_stem = nltk.stem.SnowballStemmer('english')

def clean_html(sent):
    cleaner1 = re.compile('<.*?>')
    cleaned_sent = re.sub(cleaner1,' ',sent)
    return cleaned_sent

def clean_punct(sent):
    cleaner2 = re.compile('[#|,|!|\|/|:|\'|"|;|)|(|?|*|]')
    cleaned_text = re.sub(cleaner2,' ',sent)
    return cleaned_text

print(stop_words)
print('*****************************************')
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'l
*****************************************
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'l
```

In [7]:
```
# Function to clean the text .
import string
cleaned_data =[]
all_positive_word=[]
all_negative_word=[]
i=0
s=''
str1=''
```

```
for sent in final_data['Text'].values:
    sent = clean_html(sent)
    cleaned_sent=[]
    for w in sent.split():
        for word in clean_punct(w).split():
            if((word.isalpha()) and (len(word)>2)):
                if(word.lower() not in stop):
                    s = (snow_ball_stem.stem(word.lower()))
                    cleaned_sent.append(s)
                    if((final_data['Score'].values)[i] == 'positive'):
                        all_positive_word.append(s)
                    if((final_data['Score'].values)[i] == 'negative'):
                        all_negative_word.append(s)
                else:
                    continue
            else:
                continue

    str1 = " ".join(cleaned_sent)
    cleaned_data.append(str1)
    i+=1
print(cleaned_data[100])
```

```
pros dog anyth smell bad mani easi break smaller noth artifici easi con cost dog overal great
```

4 We will do analysis for only 10k points.

In [8]:
```
final_data['Cleaned_Text'] = cleaned_data
data = final_data.sample(10000)
```

5 Bag Of Words

In [9]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
bow_data = count_vect.fit_transform(data['Cleaned_Text'])
bow_data.shape
```

Out[9]: (10000, 12511)

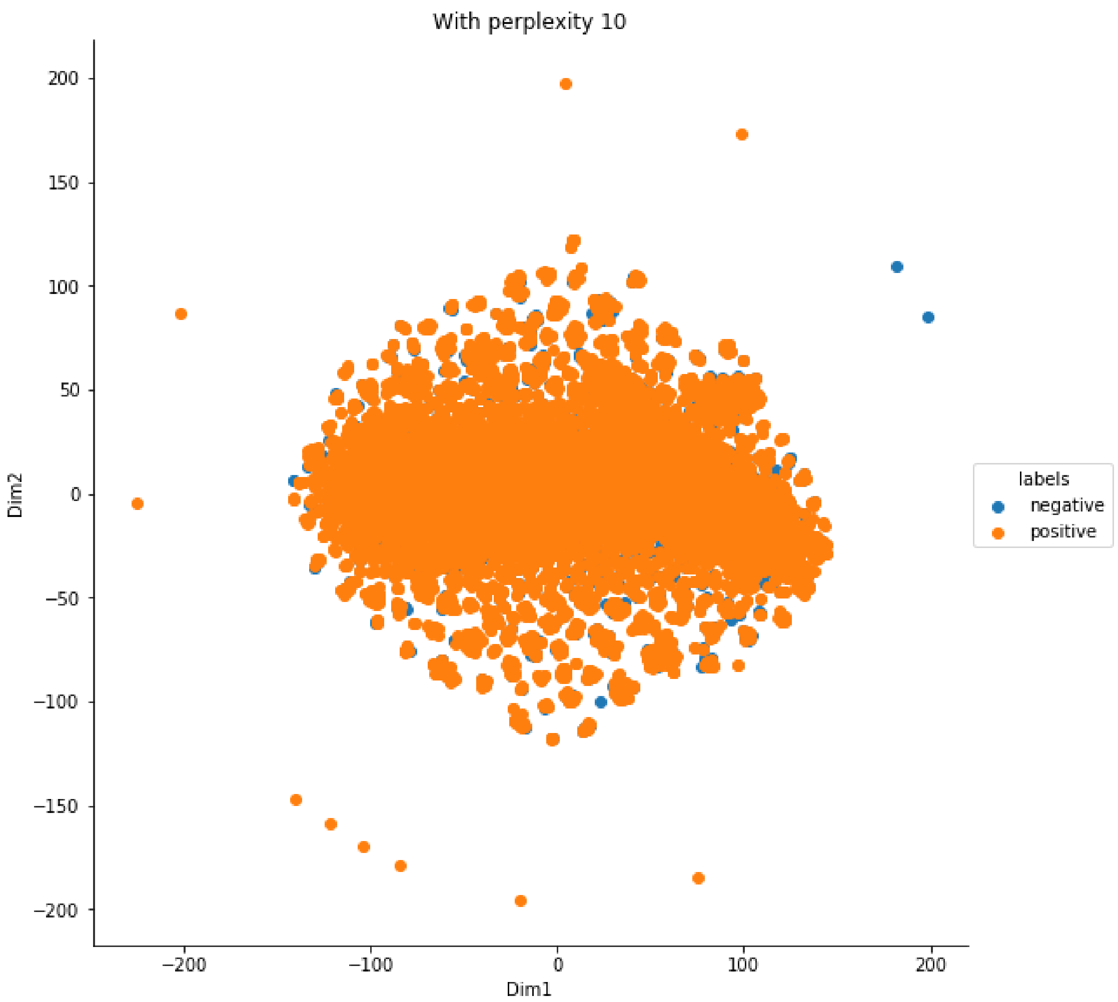
In [13]:
```
# Standarizing the data
from sklearn.preprocessing import StandardScaler
standard_bow_data = StandardScaler(with_mean=False).fit_transform(bow_data)

# Converting sparse matrix to dense matrix
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=100,algorithm='randomized',n_iter=20,random_state=42)
dense_bow_data = svd.fit_transform(standard_bow_data)
```

```
C:\Users\rites\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarn
  warnings.warn(msg, DataConversionWarning)
```

In [145]:
```
labels = data['Score'] # This stores the label of all 10k values.

from sklearn.manifold import TSNE
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

model = TSNE(n_components=2,random_state=42,perplexity=10,n_iter=3000)
bow_tsne_data1 = model.fit_transform(dense_bow_data)
bow_tsne_data1.shape
```

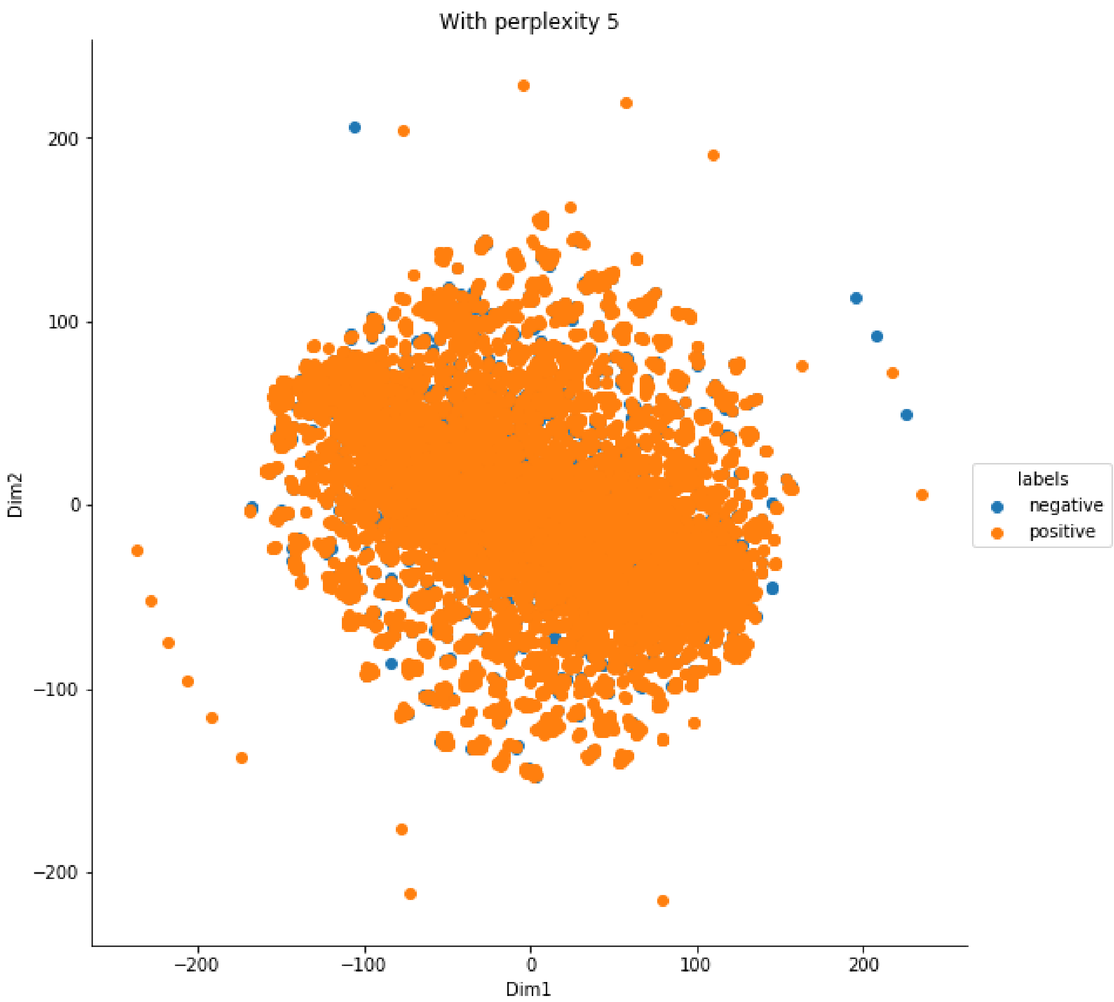
Out[145]:
```
(10000, 2)
```

In [14]:
```
labeled_data = np.vstack((bow_tsne_data1.T,labels)).T

bow_df1 = pd.DataFrame(data=labeled_data,columns=("Dim1","Dim2","labels"))

# Plotting the data
sns.FacetGrid(bow_df1,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_legend(
plt.title("With perplexity 10")
plt.show()
```

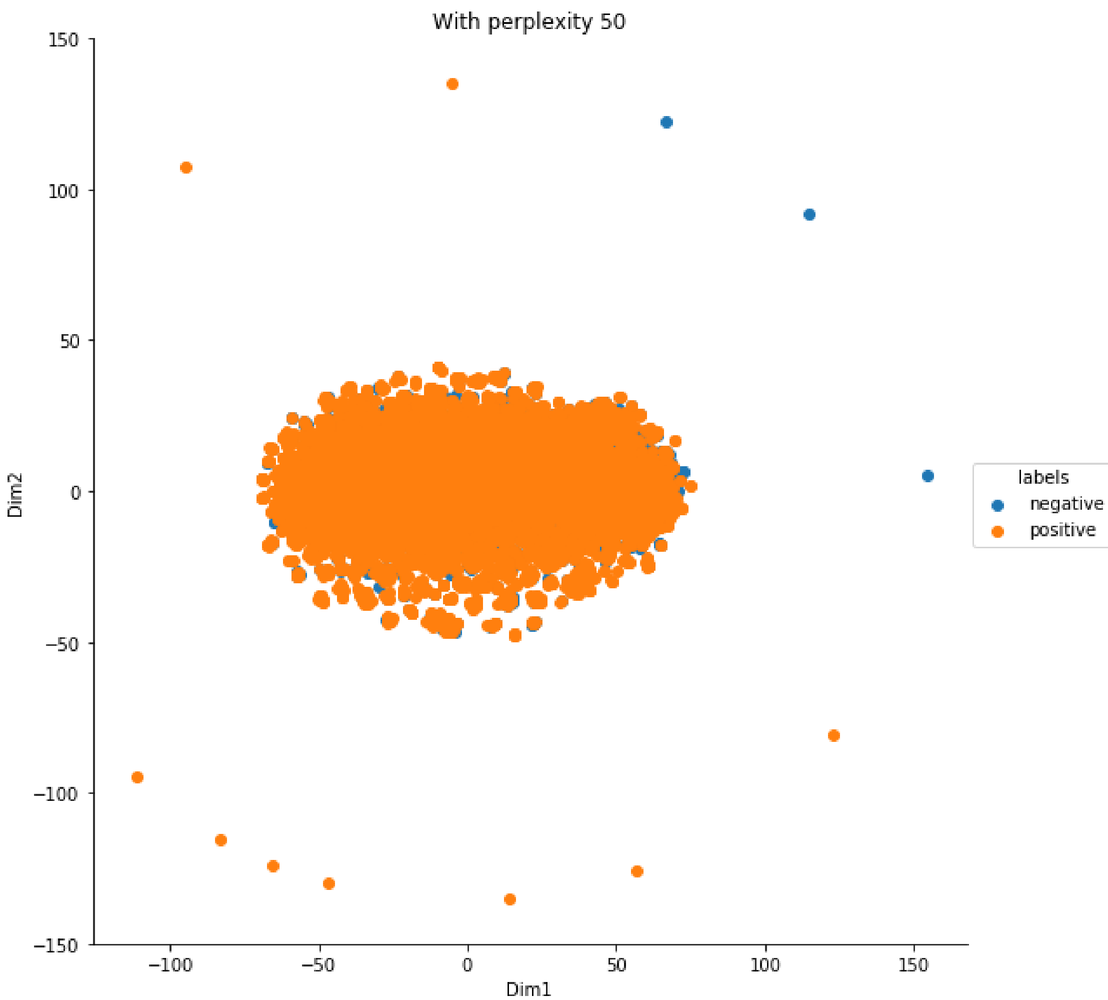


In [150]:
```
#with perplexity 5
model1 = TSNE(n_components=2,random_state=42,perplexity=5,n_iter=3000)
bow_tsne_data2 = model1.fit_transform(dense_bow_data)
```

In [16]:
```
labeled_data1 = np.vstack((bow_tsne_data2.T,labels)).T

bow_df2 = pd.DataFrame(data=labeled_data1,columns=("Dim1","Dim2","labels"))

# Plotting the data
sns.FacetGrid(bow_df2,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_legend(
plt.title("With perplexity 5")
plt.show()
```




```
In [17]: #with perplexity 50
         model2 = TSNE(n_components=2,random_state=42,perplexity=50,n_iter=3000)
         bow_tsne_data3 = model2.fit_transform(dense_bow_data)

In [18]: labeled_data2 = np.vstack((bow_tsne_data3.T,labels)).T

         bow_df3 = pd.DataFrame(data=labeled_data2,columns=("Dim1","Dim2","labels"))

         # Plotting the data
         sns.FacetGrid(bow_df3,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_legend()
         plt.title("With perplexity 50")
         plt.show()
```



6 Bi-grams

```
In [153]: bigram1 = CountVectorizer(ngram_range=(1,2))
```

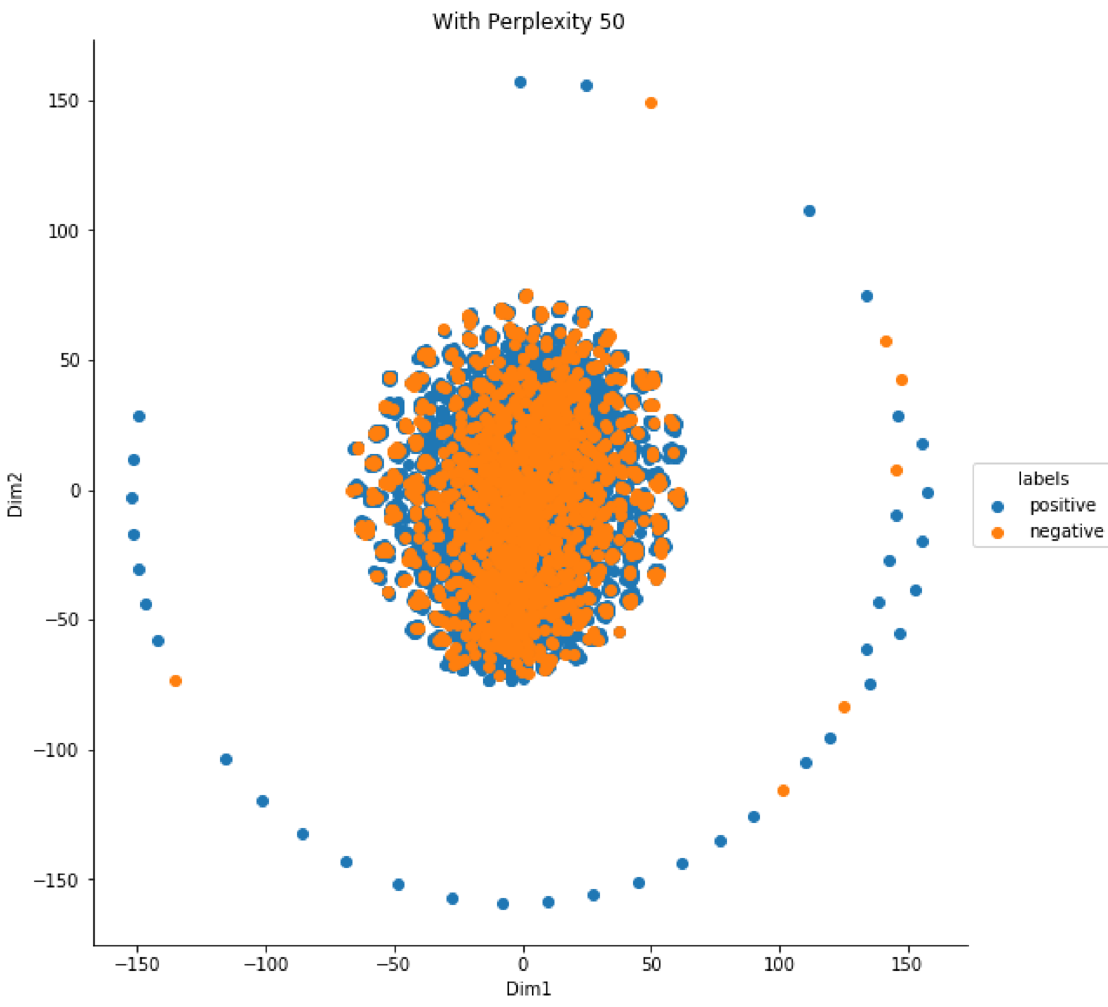
```
In [154]: bigram_data1 = bigram1.fit_transform(data['Cleaned_Text'])
          bigram_data1.shape
```

```
Out[154]: (10000, 212736)
```

```
In [164]: # standerizing data
          standard_bigram1 = StandardScaler(with_mean=False).fit_transform(bigram_data1)

          # Converting sparse matrix to dense matrix
          bigram_svd = TruncatedSVD(n_components=200,n_iter=200,algorithm='randomized',random_s
          dense_bigram1  = bigram_svd.fit_transform(standard_bigram1)
```

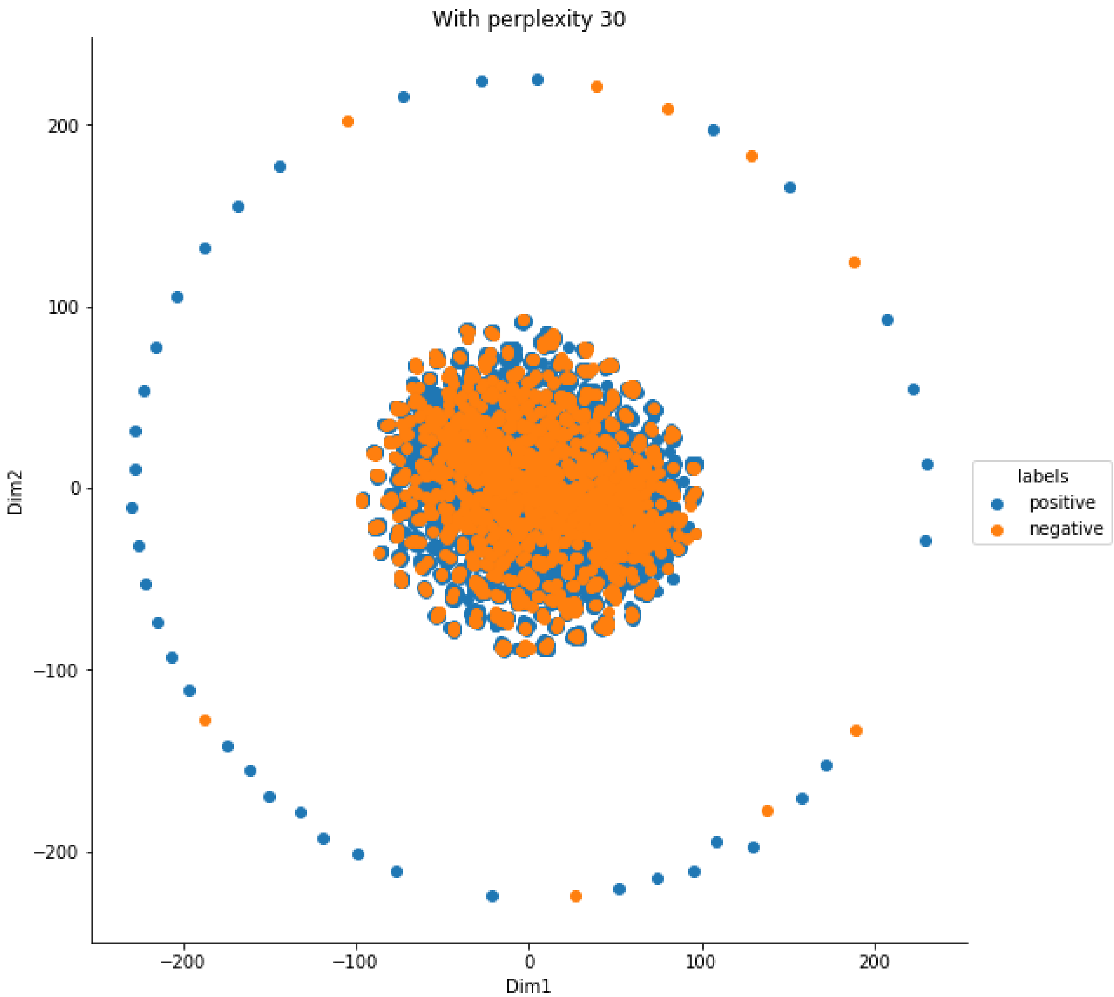
```
C:\Users\rites\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarni
  warnings.warn(msg, DataConversionWarning)
```

In [165]:
```
# Plotting tsne with perplexity 50
bigram_tsne_data1 = model2.fit_transform(dense_bigram1)

labeled_bigram1 = np.vstack((bigram_tsne_data1.T,labels)).T

bigram_df1 = pd.DataFrame(data=labeled_bigram1,columns=("Dim1","Dim2","labels"))

sns.FacetGrid(bigram_df1,hue="labels",size=8).map(plt.scatter,"Dim1","Dim2").add_lege
plt.title("With Perplexity 50")
plt.show()
```



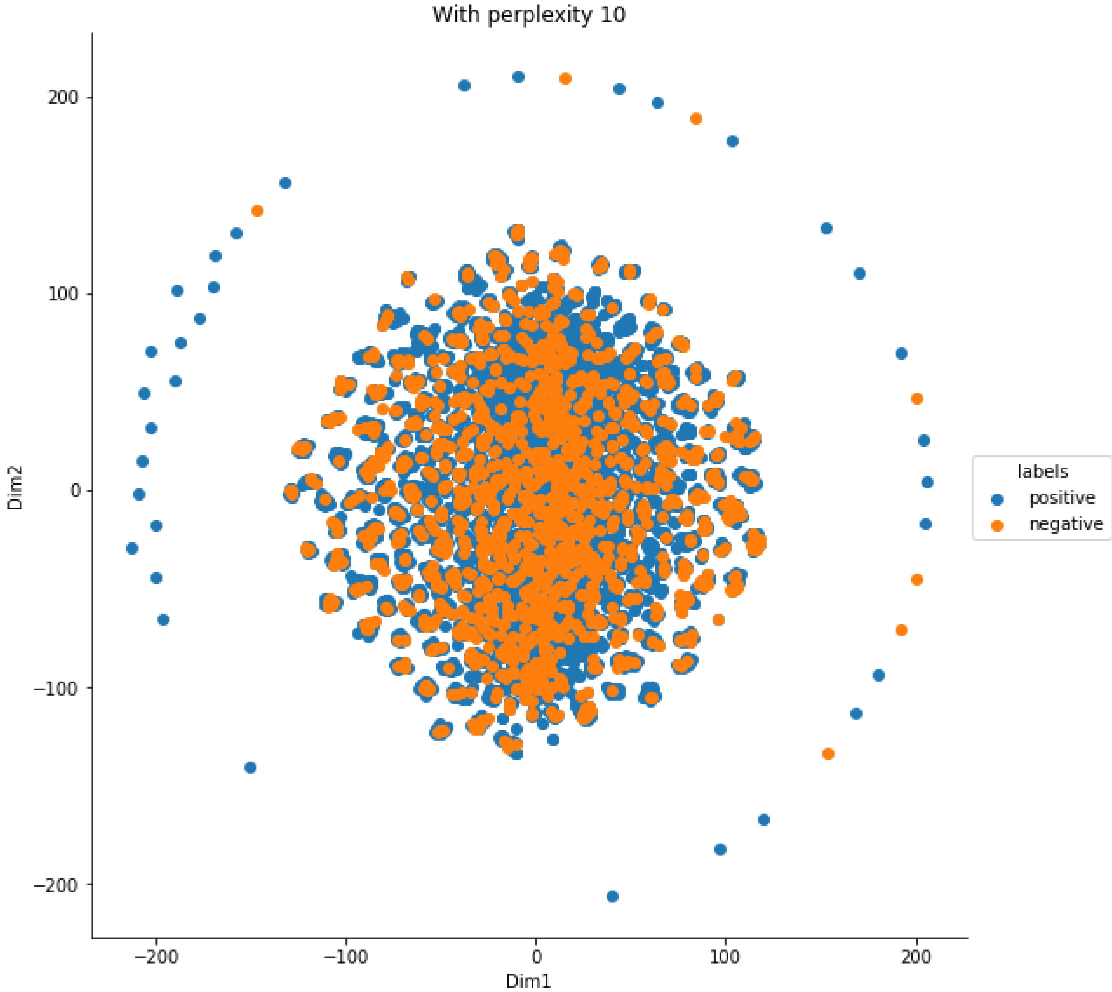
In [173]:
```
# TSNE With perplexity 30
bigram_tsne_model = TSNE(n_components=2,n_iter=5000,perplexity=30,random_state=42)
```

```
bigram_tsne_data2 = bigram_tsne_model.fit_transform(dense_bigram1)

labeled_bigram2 = np.vstack((bigram_tsne_data2.T,labels)).T

bigram_df2 = pd.DataFrame(data=labeled_bigram2,columns=("Dim1","Dim2","labels"))

sns.FacetGrid(bigram_df2,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_lege
plt.title("With perplexity 30")
plt.show()
```



In [167]:
```
# Tsne with perplexity 10
bigram_tsne_data3 = model.fit_transform(dense_bigram1)

labeled_bigram3 = np.vstack((bigram_tsne_data3.T,labels)).T
```

```
bigram_df3 = pd.DataFrame(labeled_bigram3,columns=("Dim1","Dim2","labels"))

sns.FacetGrid(bigram_df3,hue='labels',size=8).map(plt.scatter,'Dim1','Dim2').add_lege
plt.title("With perplexity 10")
plt.show()
```



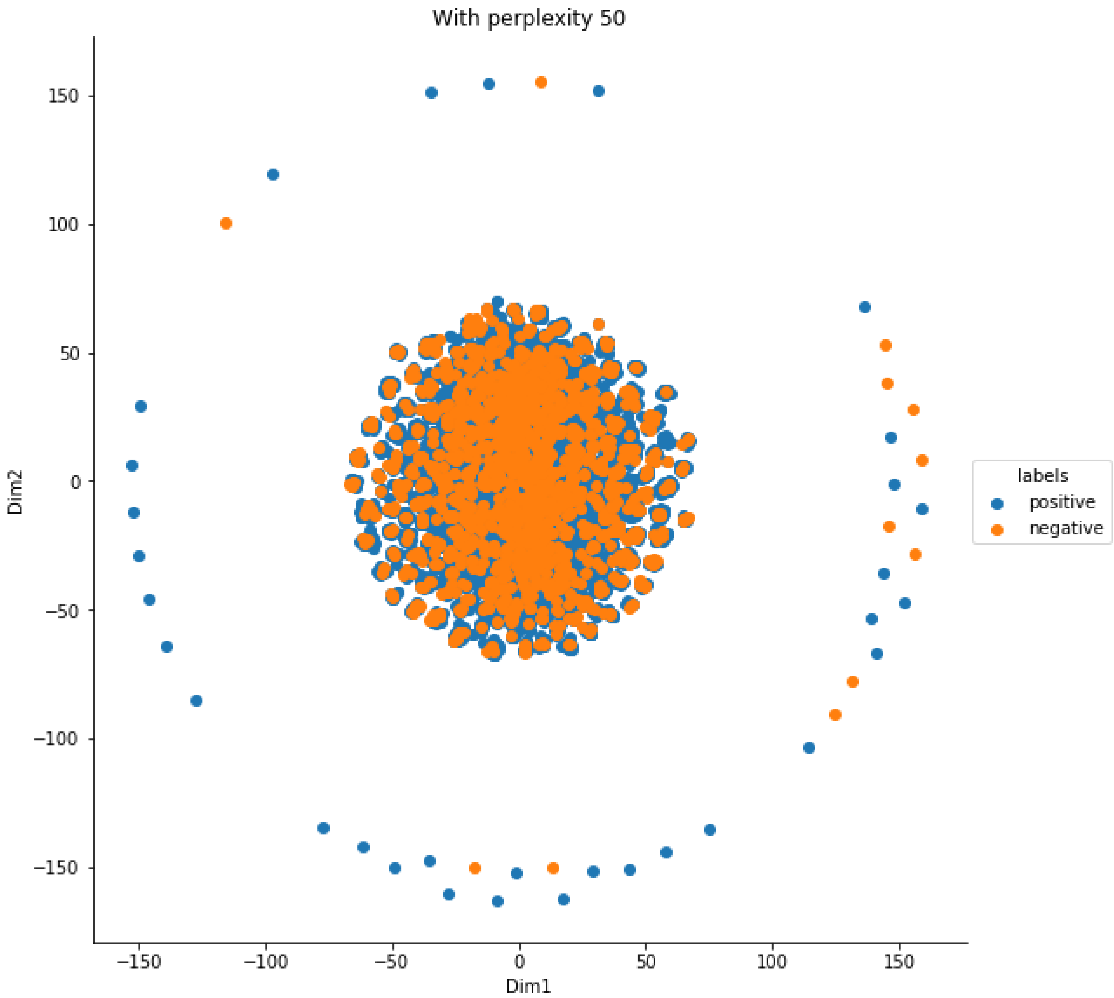
7 Bigrams and Tri-grams

```
In [168]: trigram = CountVectorizer(ngram_range=(1,3))

          trigram_data1 = trigram.fit_transform(data['Cleaned_Text'])
          trigram_data1.shape

Out[168]: (10000, 518655)

In [169]: # Standarizing the data
          standard_trigram = StandardScaler(with_mean=False).fit_transform(trigram_data1)
```

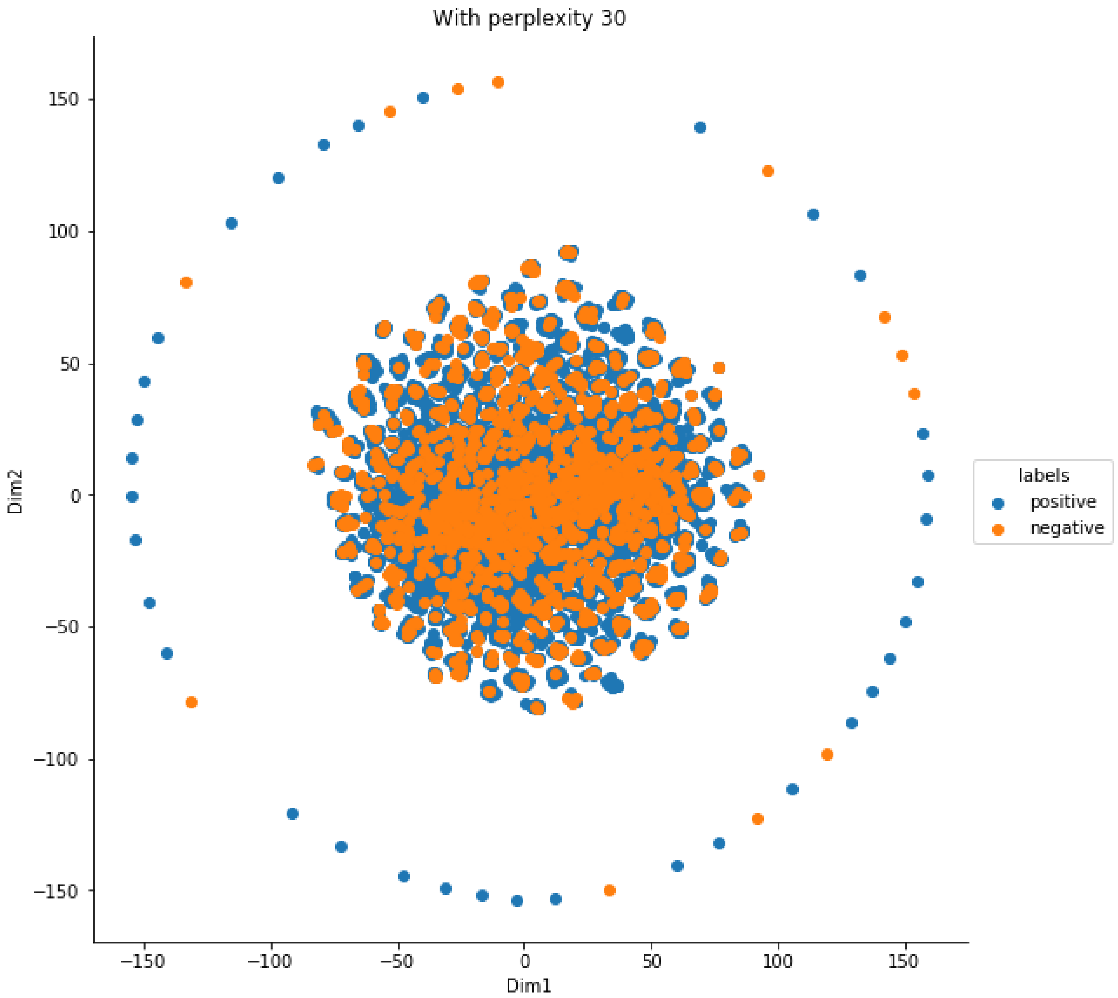
```
# Converting sparse data into dense
dense_trigram = bigram_svd.fit_transform(standard_trigram)
```

```
C:\Users\rites\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarni
  warnings.warn(msg, DataConversionWarning)
```

```
In [171]: # Tsne with perplexity 50
          trigram_tsne_data1 = model2.fit_transform(dense_trigram)

          labeled_trigram1 = np.vstack((trigram_tsne_data1.T,labels)).T

          trigram_df1 = pd.DataFrame(data=labeled_trigram1,columns=("Dim1","Dim2","labels"))

          sns.FacetGrid(trigram_df1,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_leg
          plt.title("With perplexity 50")
          plt.show()
```



```
In [172]: # Tsne with perplexity 30
          trigram_tsne_data1 = bigram_tsne_model.fit_transform(dense_trigram)

          labeled_trigram2 = np.vstack((trigram_tsne_data1.T,labels)).T

          trigram_df2 = pd.DataFrame(data=labeled_trigram2,columns=("Dim1","Dim2","labels"))

          sns.FacetGrid(trigram_df2,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_leg
          plt.title("With perplexity 30")
          plt.show()
```



8 Tf-idf

```
In [9]: tfidf1 = TfidfVectorizer(ngram_range=(1,2))
```

```
In [21]: tfidf_data = tfidf1.fit_transform(data['Cleaned_Text'])
         tfidf_data.shape
```

Out[21]: (10000, 217245)

In [26]:
```
feature_names = tfidf1.get_feature_names()

doc = 0
feature_index = tfidf_data[doc,:].nonzero()[1]

for i in feature_index:
    print(feature_names[i],tfidf_data[doc,i])
```

```
big 0.03977818493830226
fan 0.04618096784215786
crystal 0.11825910960793536
light 0.08753332315687684
beverag 0.059299248672358716
like 0.08385418166588027
lemonad 0.13331054319482313
lot 0.035958087810468074
drink 0.06864150557393292
summer 0.05729782164895767
excit 0.051833792625843526
tri 0.024735429116158454
new 0.04069196683361026
margarita 0.14036163148831898
contain 0.03949197104904636
hold 0.050652305444079816
powder 0.046718722982778
packet 0.09811205576036987
make 0.05174097572408446
quart 0.07406819352194781
mix 0.03453822269188741
thorough 0.06966244878663386
result 0.05024426712952593
odd 0.060979009689112074
murki 0.08460108750461398
green 0.04249914823826864
color 0.09664069242707456
almost 0.04005018767134891
could 0.035764995470429224
glow 0.07795557129973611
dark 0.04518827642096567
got 0.036018291796508416
past 0.048485177550367506
weird 0.059827131713318625
pour 0.0504461157064459
bring 0.05017791770712563
mouth 0.050934309694070326
thought 0.04007320677790015
```

odor 0.06703129835209942
smell 0.040004314601666376
type 0.04518827642096567
cleaner 0.07406819352194781
ammonia 0.08246170913323456
ignor 0.07072881936600448
tast 0.08691229169850105
realli 0.08566667637067926
lime 0.05847930883072138
get 0.02580262104309784
instant 0.05079227075137091
artifici 0.05037835957092256
sweeten 0.04815830089223952
aspartam 0.07018081574415949
thing 0.036325116485403076
notic 0.04681135610217144
textur 0.04235187263457081
soft 0.04640137877390421
left 0.0495385172704125
unpleas 0.06665527159741157
coat 0.05265373246748086
put 0.03744851160960396
pitcher 0.07406819352194781
fridg 0.05947200025191357
sever 0.041512784687361665
hour 0.0488235933867072
hope 0.08733112543218115
would 0.05411374359125541
better 0.032579079644906486
cold 0.050934309694070326
disappoint 0.04571319442300511
cocktail 0.07018081574415949
lowcal 0.09124660370949184
altern 0.048485177550367506
tradit 0.053272492523190974
guess 0.04966303140067214
stick 0.04618096784215786
origin 0.04832034621353728
big fan 0.0601959253833383
fan crystal 0.09124660370949184
crystal light 0.13406259670419884
light beverag 0.09124660370949184
beverag like 0.08735922593170352
like lemonad 0.08246170913323456
lemonad lot 0.09124660370949184
lot drink 0.08460108750461398
drink summer 0.09124660370949184
summer excit 0.09124660370949184

excit tri 0.06917067672347885
tri new 0.06703129835209942
new margarita 0.09124660370949184
margarita contain 0.09124660370949184
contain hold 0.08735922593170352
hold powder 0.09124660370949184
powder packet 0.08460108750461398
packet packet 0.09124660370949184
packet make 0.07682633194903735
make quart 0.08735922593170352
quart mix 0.09124660370949184
mix thorough 0.09124660370949184
thorough result 0.09124660370949184
result drink 0.09124660370949184
drink odd 0.09124660370949184
odd murki 0.09124660370949184
murki green 0.09124660370949184
green color 0.07682633194903735
color almost 0.09124660370949184
almost could 0.09124660370949184
could glow 0.09124660370949184
glow dark 0.09124660370949184
dark got 0.09124660370949184
got past 0.08735922593170352
past weird 0.09124660370949184
weird color 0.09124660370949184
color pour 0.09124660370949184
pour bring 0.09124660370949184
bring mouth 0.09124660370949184
mouth thought 0.09124660370949184
thought odor 0.09124660370949184
odor smell 0.09124660370949184
smell like 0.057437786956248765
like type 0.08735922593170352
type cleaner 0.09124660370949184
cleaner ammonia 0.09124660370949184
ammonia like 0.09124660370949184
like ignor 0.09124660370949184
ignor tast 0.09124660370949184
tast realli 0.06436951705038253
realli tast 0.062288421249204376
tast lime 0.08460108750461398
lime get 0.08735922593170352
get instant 0.09124660370949184
instant tast 0.08246170913323456
tast artifici 0.08460108750461398
artifici sweeten 0.06560832848826076
sweeten aspartam 0.09124660370949184

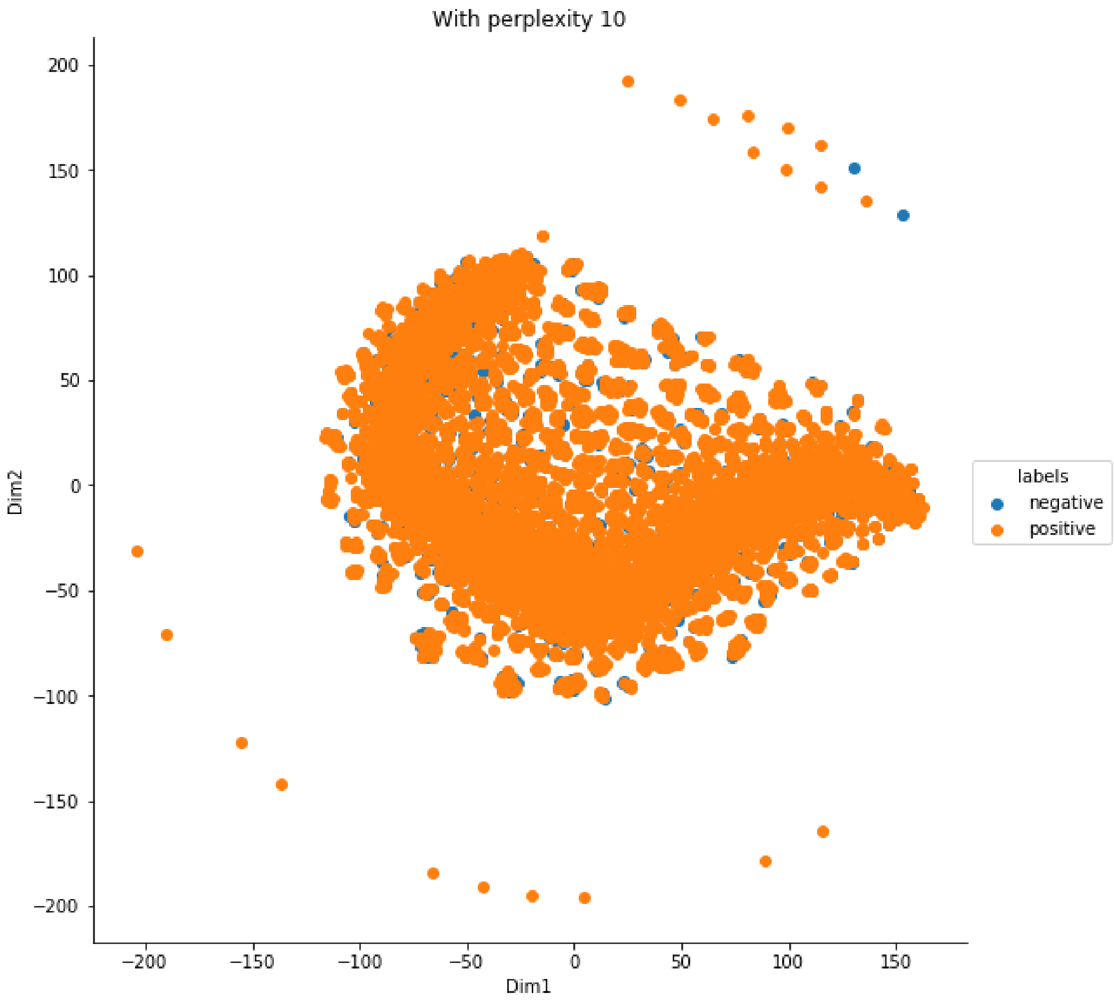
```
aspartam thing 0.09124660370949184
thing realli 0.07682633194903735
realli notic 0.07682633194903735
notic textur 0.09124660370949184
textur soft 0.08246170913323456
soft left 0.09124660370949184
left unpleas 0.09124660370949184
unpleas coat 0.09124660370949184
coat put 0.09124660370949184
put pitcher 0.09124660370949184
pitcher fridg 0.09124660370949184
fridg sever 0.09124660370949184
sever hour 0.07330078780228942
hour hope 0.09124660370949184
hope would 0.13188952975447313
would tast 0.06496892805154764
tast better 0.05772400084550468
better cold 0.08735922593170352
cold realli 0.09124660370949184
realli disappoint 0.08460108750461398
disappoint like 0.09124660370949184
like margarita 0.08735922593170352
margarita cocktail 0.09124660370949184
cocktail hope 0.09124660370949184
would make 0.06276789381962325
make lowcal 0.09124660370949184
lowcal altern 0.09124660370949184
altern tradit 0.08460108750461398
tradit guess 0.09124660370949184
guess stick 0.08735922593170352
stick origin 0.08735922593170352
origin crystal 0.08735922593170352
light lemonad 0.08460108750461398
```

In [14]:
```
# Standarizing data
standard_tfidf_data1 = StandardScaler(with_mean=False).fit_transform(tfidf_data)

# coneverting sparse matrix into dense matrix
svd = TruncatedSVD(n_components=100,n_iter=100,algorithm='randomized',random_state=42)
dense_tfidf_data=svd.fit_transform(standard_tfidf_data1)
dense_tfidf_data.shape
```

Out[14]:
```
(10000, 100)
```

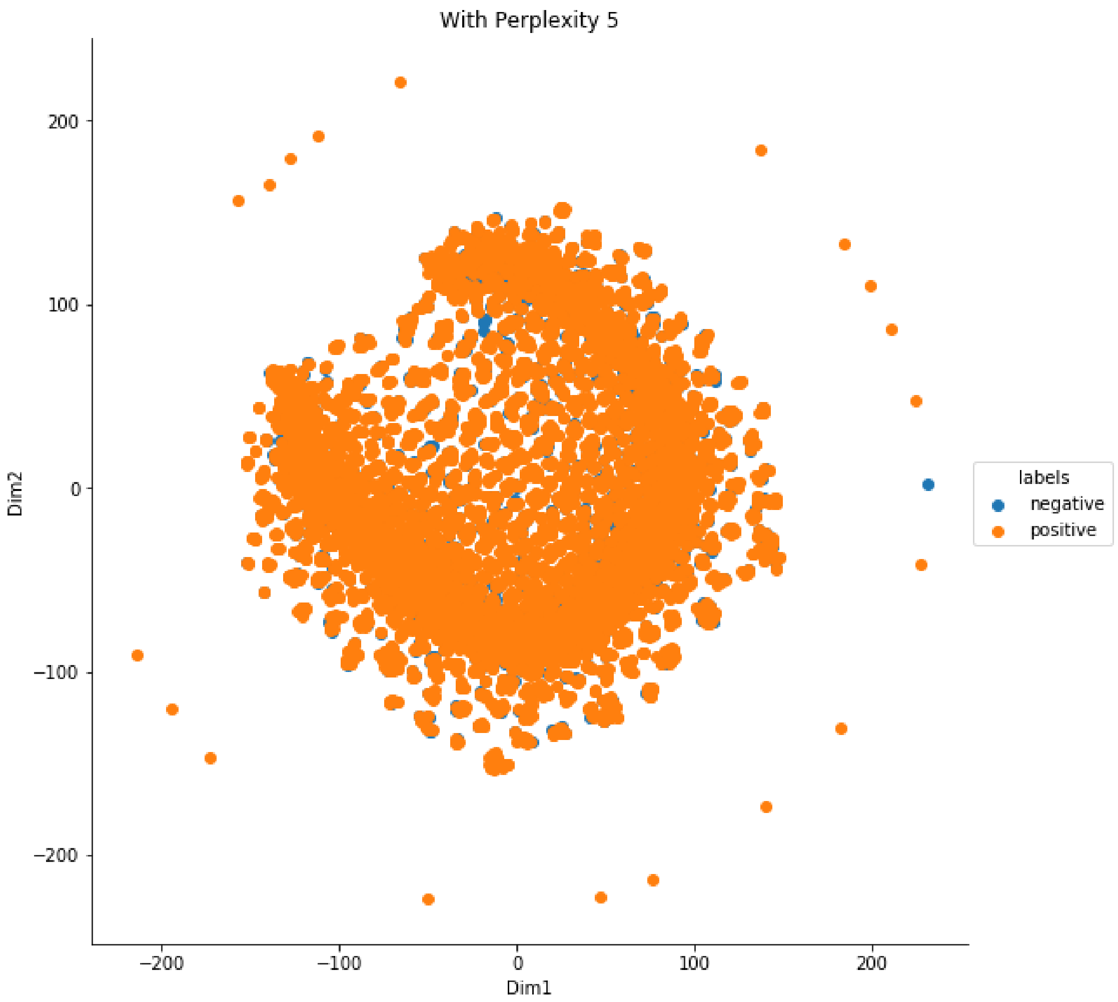
In [23]:
```
# Reducing 100 dimension to 2 dimension using tsne model
tfidf_tsne_data1 = model.fit_transform(dense_tfidf_data)
tfidf_tsne_data1.shape
```

```
tfidf_labeled_data1 = np.vstack((tfidf_tsne_data1.T,labels)).T

tfidf_df1 = pd.DataFrame(data=tfidf_labeled_data1,columns=('Dim1','Dim2','labels'))

sns.FacetGrid(tfidf_df1,hue='labels',size=8).map(plt.scatter,'Dim1','Dim2').add_legend
plt.title("With perplexity 10")
plt.show()
```

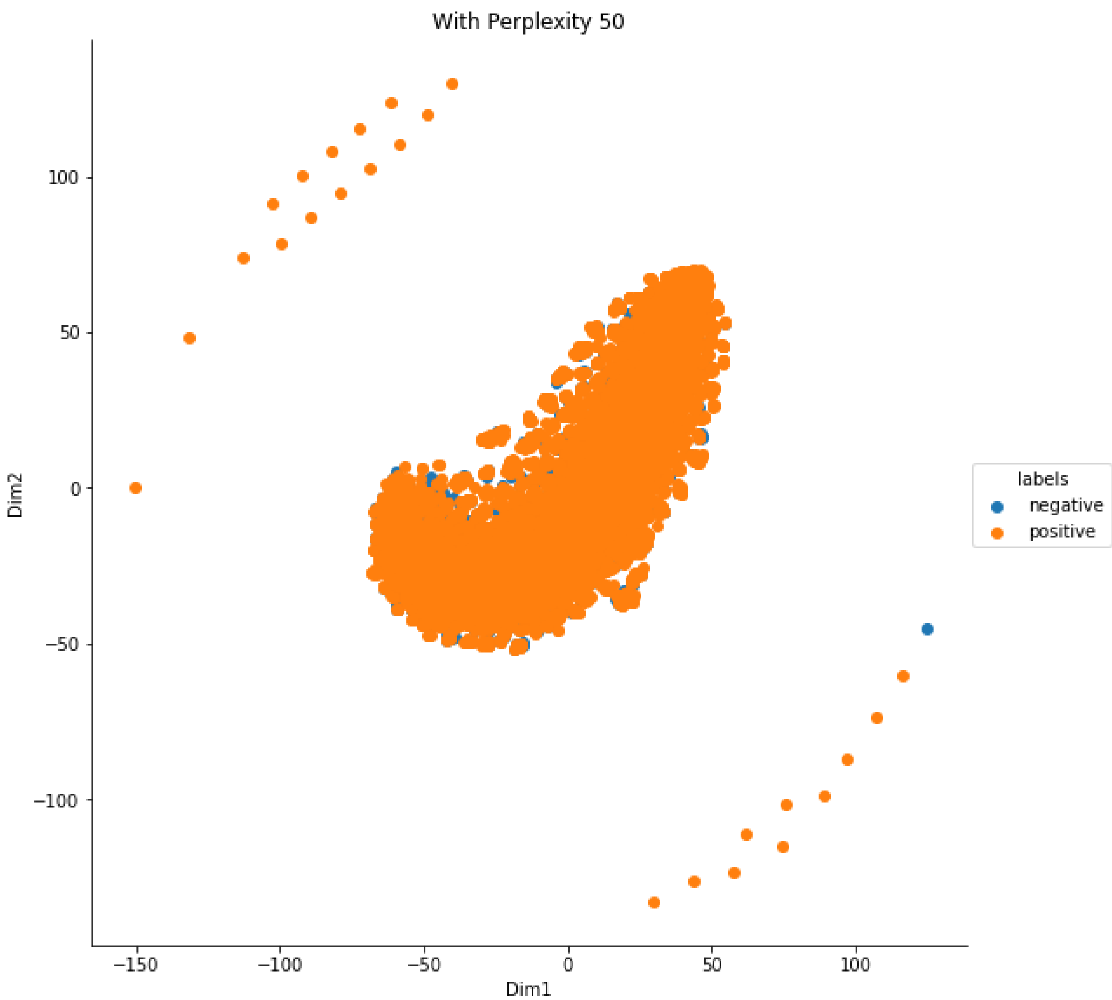



```
In [24]: # Using model of tsne with perplexity 5
         tfidf_tsne_data2 = model1.fit_transform(dense_tfidf_data)

         labeled_tfidf_data2 = np.vstack((tfidf_tsne_data2.T,labels)).T

         tfidf_df2 = pd.DataFrame(data=labeled_tfidf_data2,columns=("Dim1","Dim2","labels"))
```

```
sns.FacetGrid(tfidf_df2,hue='labels',size=8).map(plt.scatter,'Dim1','Dim2').add_legenc
plt.title("With Perplexity 5")
plt.show()
```



In [25]:
```
# Tsne with perplexity 50
tfidf_tsne_data3 = model2.fit_transform(dense_tfidf_data)

tfidf_labeled_data3 = np.vstack((tfidf_tsne_data3.T,labels)).T

tfidf_df3 = pd.DataFrame(data=tfidf_labeled_data3,columns=("Dim1","Dim2","labels"))

sns.FacetGrid(tfidf_df3,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_legenc
plt.title("With Perplexity 50")
plt.show()
```



9 Avg Word2Vec

```
In [144]: from gensim.models import Word2Vec
          from gensim.models import KeyedVectors

          model_w2v = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin',bi

In [139]: model_w2v.wv.similarity('king','queen')

C:\Users\rites\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning: Call to
  """Entry point for launching an IPython kernel.
C:\Users\rites\Anaconda3\lib\site-packages\gensim\matutils.py:737: FutureWarning: Conversion of
  if np.issubdtype(vec.dtype, np.int):


Out[139]: 0.6510957
```

In [127]:
```python
import gensim
list_of_sent=[]
for sent in final_data['Text'].values:
    filtered_sentence=[]
    sent = clean_html(sent)
    for w in sent.split():
        for cleaned_word in clean_punct(w).split():
            if(cleaned_word.isalpha()):
                filtered_sentence.append(cleaned_word)
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

In [74]:
```python
import gensim
my_model = gensim.models.Word2Vec(list_of_sent,min_count=5,size=50,workers=4)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-74-3b252c3d644a> in <module>()
      1 import gensim
----> 2 my_model = gensim.models.Word2Vec(list_of_sent,min_count=1,size=50,workers=4)


NameError: name 'list_of_sent' is not defined
```

In [129]:
```python
my_model.wv.most_similar('tasty')
```

```
C:\Users\rites\Anaconda3\lib\site-packages\gensim\matutils.py:737: FutureWarning: Conversion o
  if np.issubdtype(vec.dtype, np.int):
```

Out[129]:
```
[('yummy', 0.8695557117462158),
 ('delicious', 0.8563160300254822),
 ('tastey', 0.8515514135360718),
 ('satisfying', 0.8366239070892334),
 ('filling', 0.8174551725387573),
 ('flavorful', 0.792838454246521),
 ('nutritious', 0.7622948884963989),
 ('hearty', 0.7587249279022217),
 ('hardy', 0.7475316524505615),
 ('versatile', 0.7446770668029785)]
```

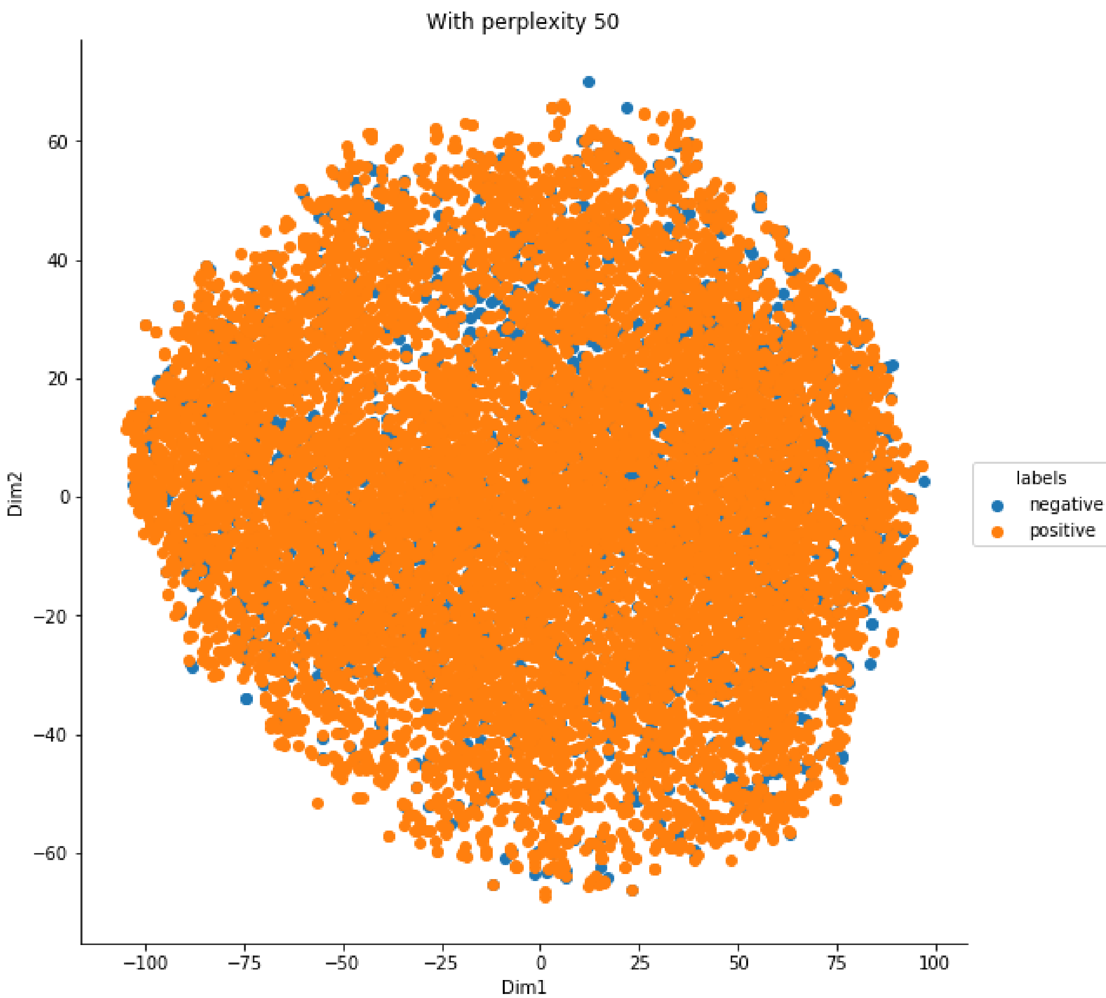
In [132]:
```python
## Calculating average word2vec for each 10k reviews

import numpy as np
```

```
list_of_w2v_sent=[]
for sent in data['Cleaned_Text'].values:
    sent_vect = np.zeros(50)
    cnt_words = 0
    for w in sent:
        try:
            vec = my_model.wv[w]
            sent_vect += vec
            cnt_words +=1
        except:
            pass
    sent_vect /= cnt_words
    list_of_w2v_sent.append(sent_vect)

print(len(list_of_w2v_sent))
print(len(list_of_w2v_sent[0]))
```

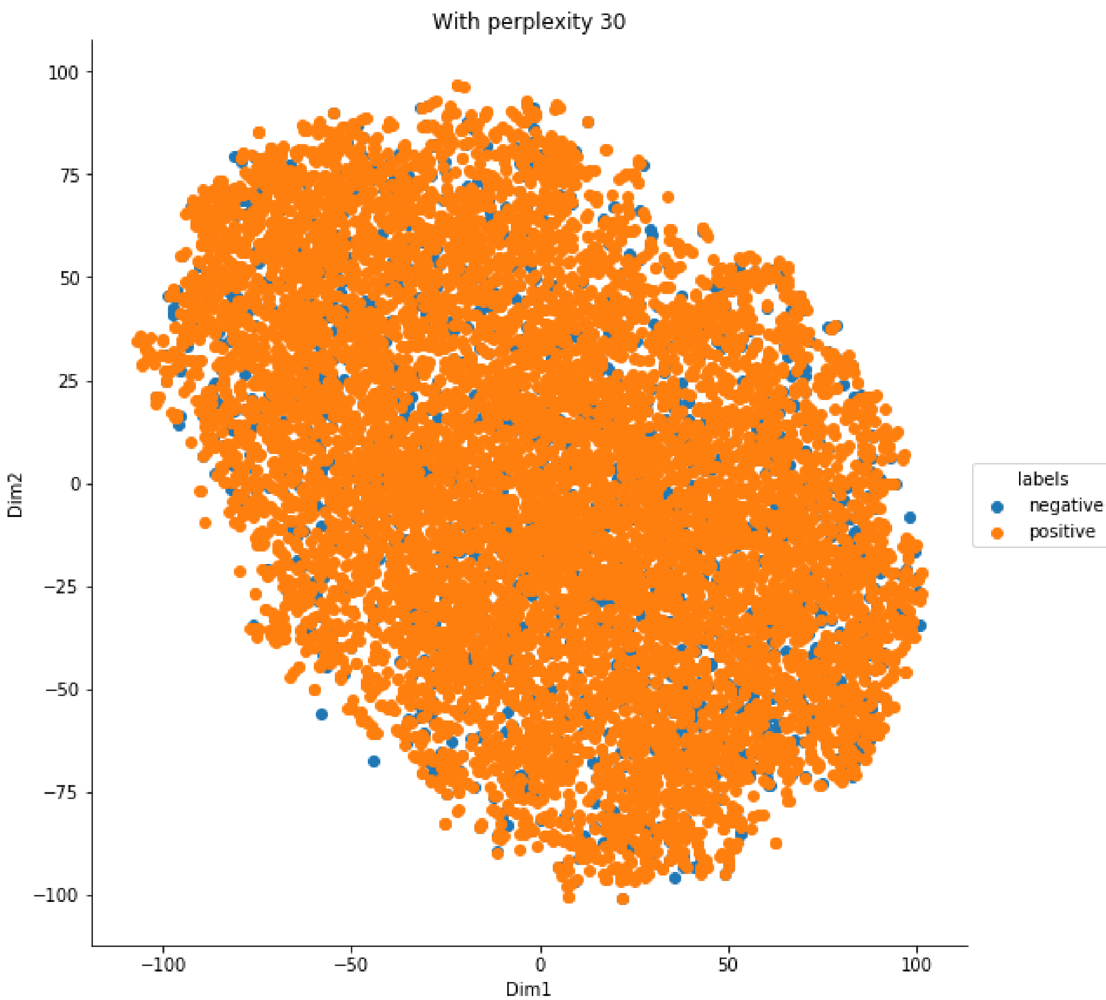
```
10000
50
```

In [133]:
```
# Tsne for avg word2vec with perplexity = 50
w2v_tsne = model2.fit_transform(list_of_w2v_sent)
```

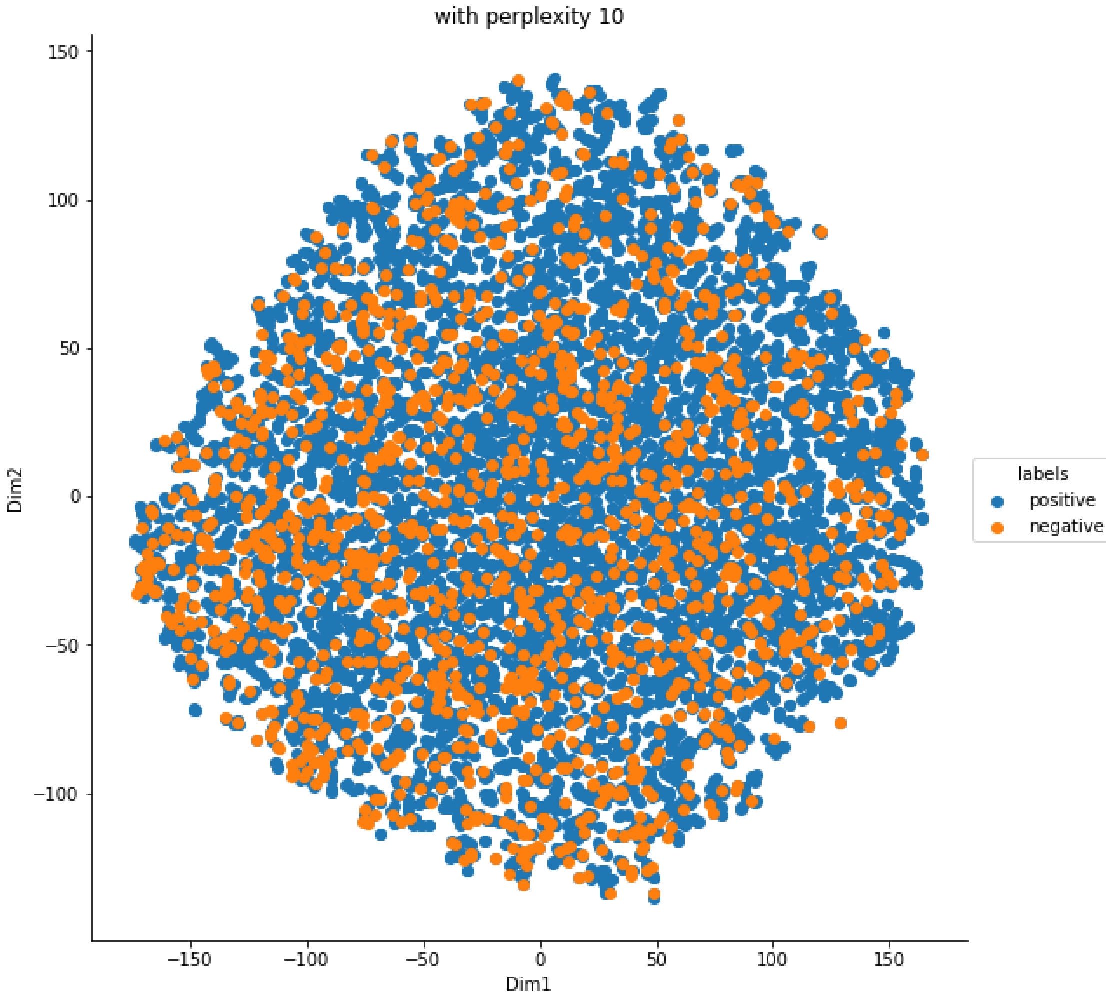
In [134]:
```
labeled_w2v1 = np.vstack((w2v_tsne.T,labels)).T

w2v_df1 = pd.DataFrame(data=labeled_w2v1,columns=("Dim1","Dim2","labels"))

sns.FacetGrid(w2v_df1,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_legend
plt.title("With perplexity 50")
plt.show()
```



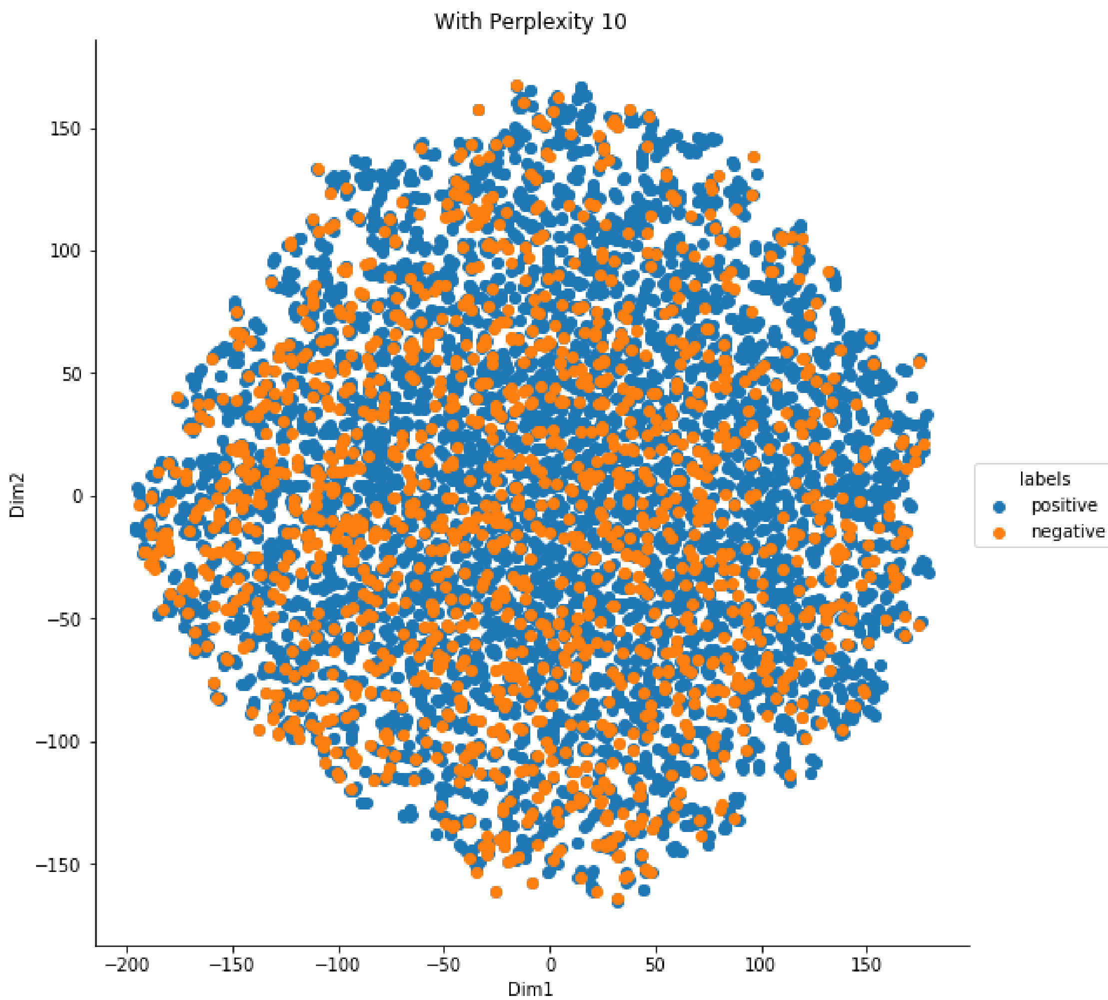

```
In [135]: # With perplexity 30
          model3 = TSNE(n_components=2,n_iter=3000,random_state=42,perplexity=30)

          w2v_tsne2 = model3.fit_transform(list_of_w2v_sent)

In [137]: labeled_w2v2 = np.vstack((w2v_tsne2.T,labels)).T

          w2v_df2 = pd.DataFrame(data=labeled_w2v2,columns=("Dim1","Dim2","labels"))

          sns.FacetGrid(w2v_df2,hue='labels',size=8).map(plt.scatter,'Dim1','Dim2').add_legend
          plt.title("With perplexity 30")
          plt.show()
```



In [142]:
```
# With perplexity 10
w2v_tsne_data3 = model.fit_transform(list_of_w2v_sent)

labeled_w2v3 = np.vstack((w2v_tsne_data3.T,labels)).T

w2v_df3 = pd.DataFrame(data=labeled_w2v3,columns=("Dim1","Dim2","labels"))

sns.FacetGrid(w2v_df3,hue='labels',size=8).map(plt.scatter,'Dim1','Dim2').add_legend
plt.title("with perplexity 10")
plt.show()
```




```
In [152]: # with perplexity 10
          w2v_tsne_data4 = model1.fit_transform(list_of_w2v_sent)

          labeled_w2v4 = np.vstack((w2v_tsne_data4.T,labels)).T

          w2v_df4 = pd.DataFrame(data=labeled_w2v4,columns=("Dim1","Dim2","labels"))

          sns.FacetGrid(w2v_df4,hue='labels',size=8).map(plt.scatter,'Dim1','Dim2').add_legend
          plt.title("With Perplexity 10")
          plt.show()
```



10 Tfidf Weighted Word2Vec

```
In [41]: # Training our own word2vec model besed on cleaned data
         list_of_w2v_sent1=[]
         for sent in final_data['Cleaned_Text'].values:
             filtered_sentence2=[]
             sent2 = clean_html(sent)
             for w in sent2.split():
                 for cleaned_word2 in clean_punct(w).split():
                     if(cleaned_word2.isalpha()):
                         filtered_sentence2.append(cleaned_word2)
                     else:
                         continue
             list_of_w2v_sent1.append(filtered_sentence2)
```

```
In [42]: len(list_of_w2v_sent1)
```

Out[42]: 364171

In [43]:
```
import gensim
my_model1 = gensim.models.Word2Vec(list_of_w2v_sent1,min_count=5,size=50,workers=4)
```

In [44]:
```
tfidf5 = TfidfVectorizer(ngram_range=(1,1))
tfidf_matrix = tfidf5.fit_transform(data['Cleaned_Text'])
tfidf_matrix.shape
```

Out[44]: (10000, 12644)

In [49]:
```
import numpy as np
# Getting Feature Names
feature_names = tfidf5.get_feature_names()

avg_tfidf_w2v = []
# Getting index of features where value is not zero
for i in range(10000):
    feature_index = tfidf_matrix[i,:].nonzero()[1]
    sum_tfidf_w2v=np.zeros(50)
    k=0
    for j in feature_index:
        try:
            k +=1
            sum_tfidf_w2v += my_model1.wv[feature_names[j]]*tfidf_matrix[i,j]
        except:
            pass
    avg_tfidf_w2v.append(sum_tfidf_w2v/k)

print(len(avg_tfidf_w2v))
```

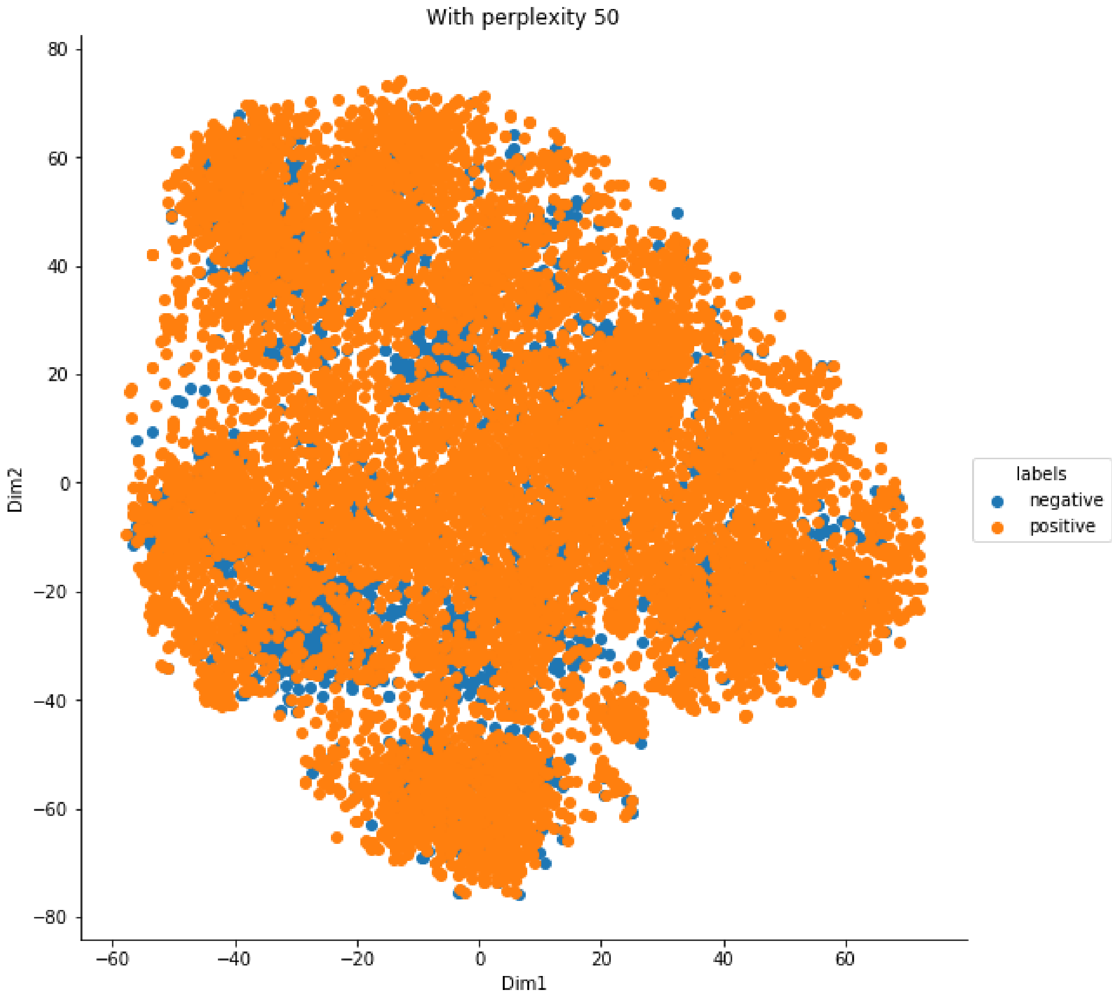
```
10000
```

In [51]:
```
# Plotting tsne with perplexity 50
from sklearn.manifold import TSNE
tsne_model1 = TSNE(n_components=2,n_iter=3500,perplexity=50,random_state=42)
tfidf_w2v_tsne_data1 = tsne_model1.fit_transform(avg_tfidf_w2v)
tfidf_w2v_tsne_data1.shape
```

Out[51]: (10000, 2)

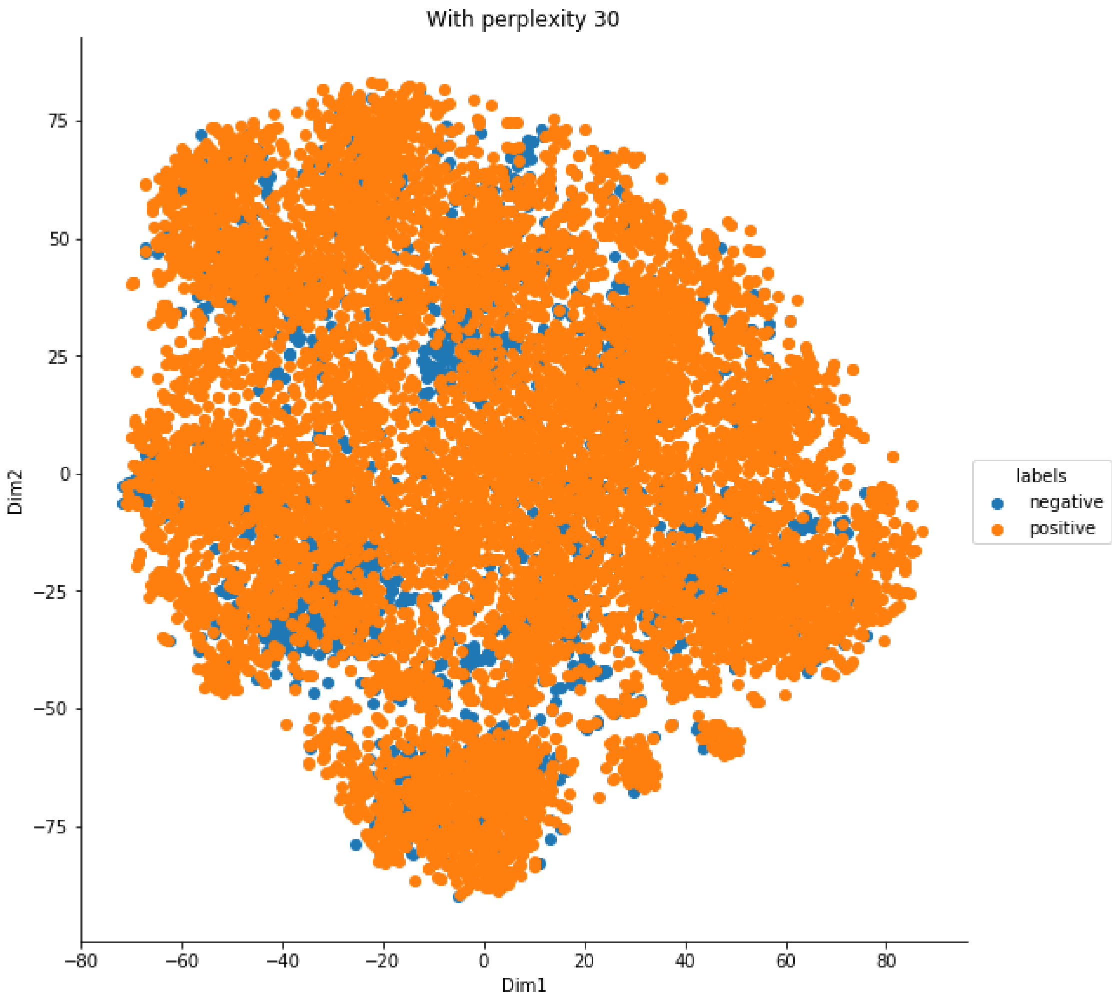
In [53]:
```
data_labels = data['Score']

# Labelling data generated by tsne
labeled_tsne_data1 = np.vstack((tfidf_w2v_tsne_data1.T,data_labels)).T
```

```
# Creating dataframe
tfidf_w2v_df1 = pd.DataFrame(data=labeled_tsne_data1,columns=("Dim1","Dim2","labels"))

# Plotting the tsne plot with perplexity 50
sns.FacetGrid(tfidf_w2v_df1,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_le
plt.title("With perplexity 50")
plt.show()
```




```
In [54]: # Plotting TSNE with perplexity 30
         tsne_model2 = TSNE(n_components=2,n_iter=2000,perplexity=30,random_state=42)
         tfidf_w2v_tsne_data2 = tsne_model2.fit_transform(avg_tfidf_w2v)
         tfidf_w2v_tsne_data2.shape

Out[54]: (10000, 2)

In [55]: # Labelling data generated by tsne
         labeled_tsne_data2 = np.vstack((tfidf_w2v_tsne_data2.T,data_labels)).T
```

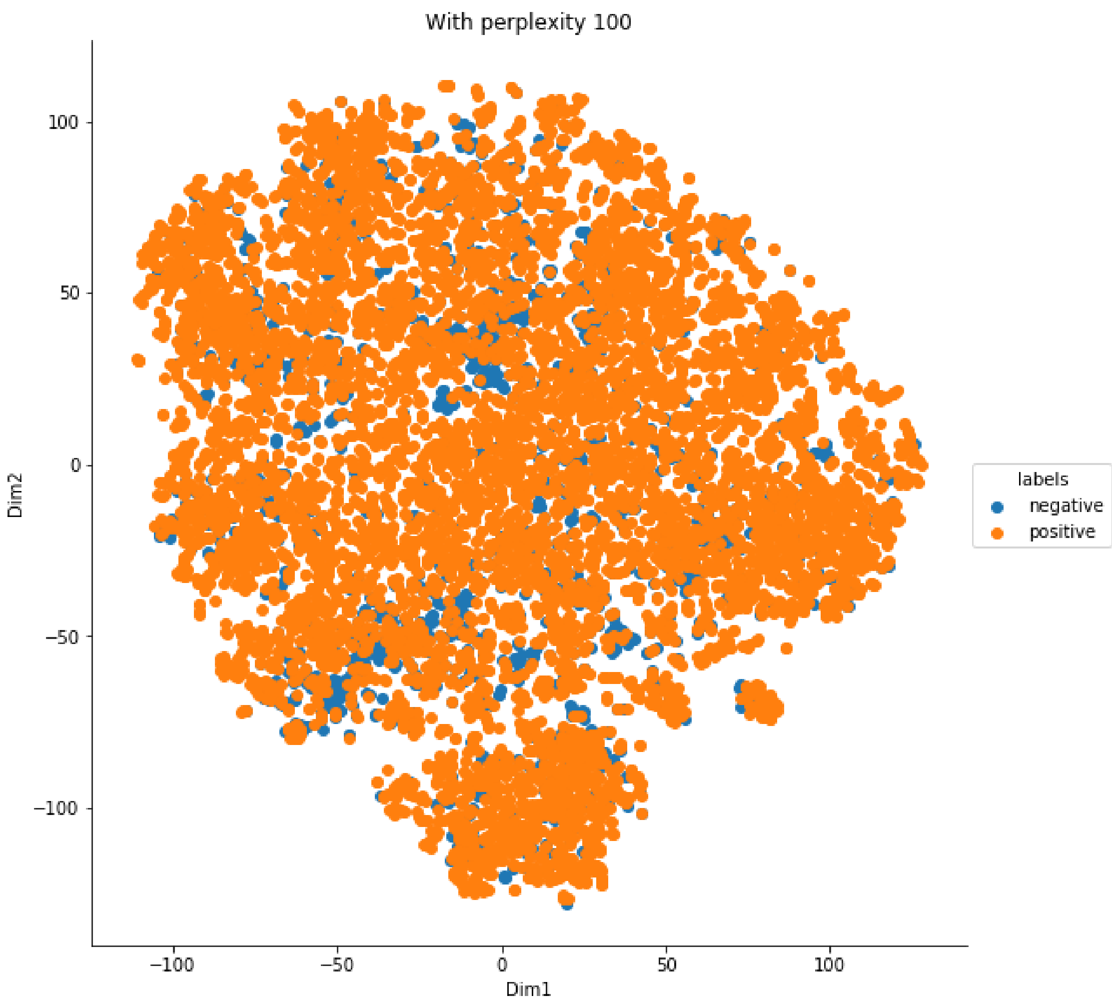
```
# Creating dataframe
tfidf_w2v_df2 = pd.DataFrame(data=labeled_tsne_data2,columns=("Dim1","Dim2","labels"))

# Plotting the tsne plot with perplexity 50
sns.FacetGrid(tfidf_w2v_df2,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_le
plt.title("With perplexity 30")
plt.show()
```



In [56]:
```
# Plotting TSNE with perplexity 10
tsne_model3 = TSNE(n_components=2,n_iter=2000,perplexity=10,random_state=42)
tfidf_w2v_tsne_data3 = tsne_model3.fit_transform(avg_tfidf_w2v)
tfidf_w2v_tsne_data3.shape
```

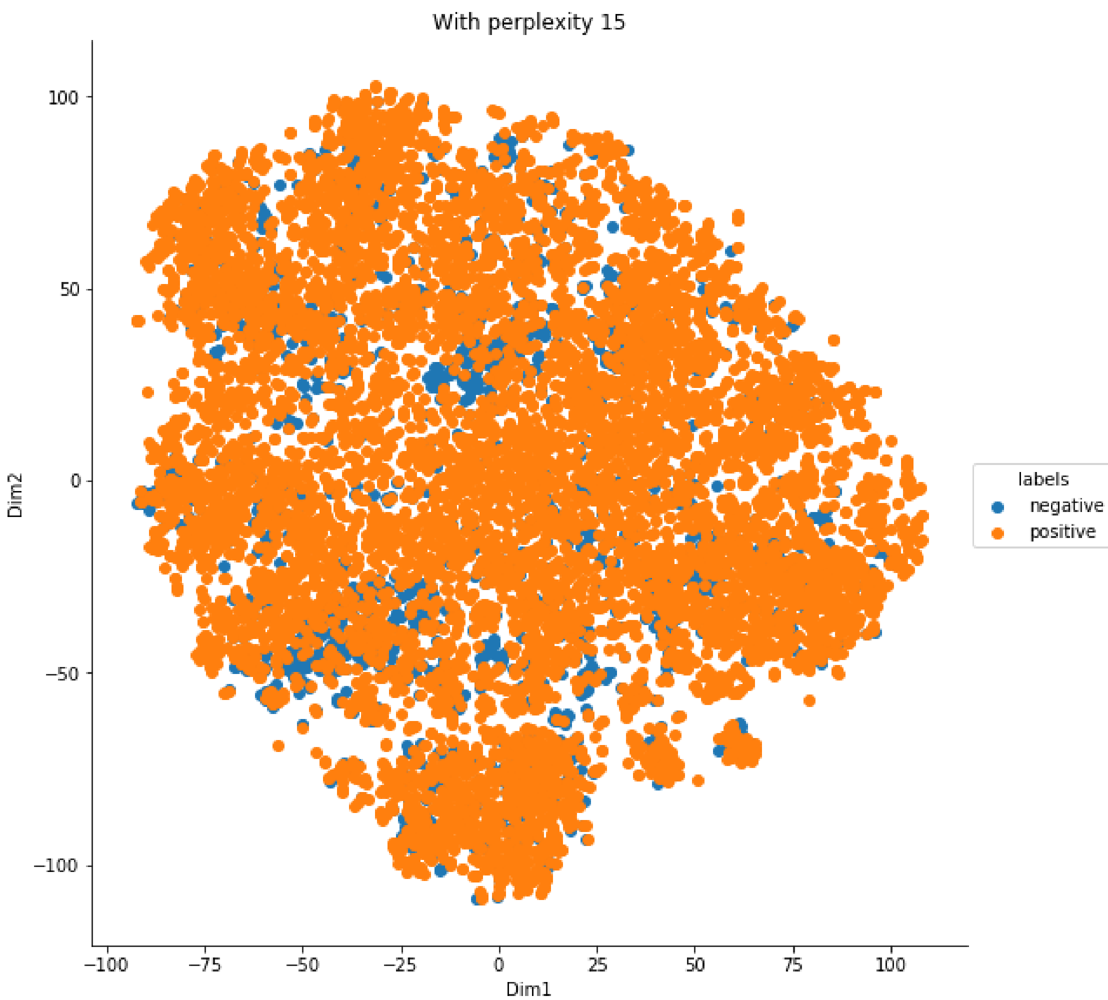
Out[56]: (10000, 2)

```
In [57]: # Labelling data generated by tsne
         labeled_tsne_data3 = np.vstack((tfidf_w2v_tsne_data3.T,data_labels)).T

         # Creating dataframe
         tfidf_w2v_df3 = pd.DataFrame(data=labeled_tsne_data3,columns=("Dim1","Dim2","labels"))

         # Plotting the tsne plot with perplexity 50
         sns.FacetGrid(tfidf_w2v_df3,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_le
         plt.title("With perplexity 10")
         plt.show()
```




```
In [58]: # Plotting TSNE with perplexity 15
         tsne_model4 = TSNE(n_components=2,n_iter=2000,perplexity=15,random_state=42)
         tfidf_w2v_tsne_data4 = tsne_model4.fit_transform(avg_tfidf_w2v)
         tfidf_w2v_tsne_data4.shape

Out[58]: (10000, 2)
```

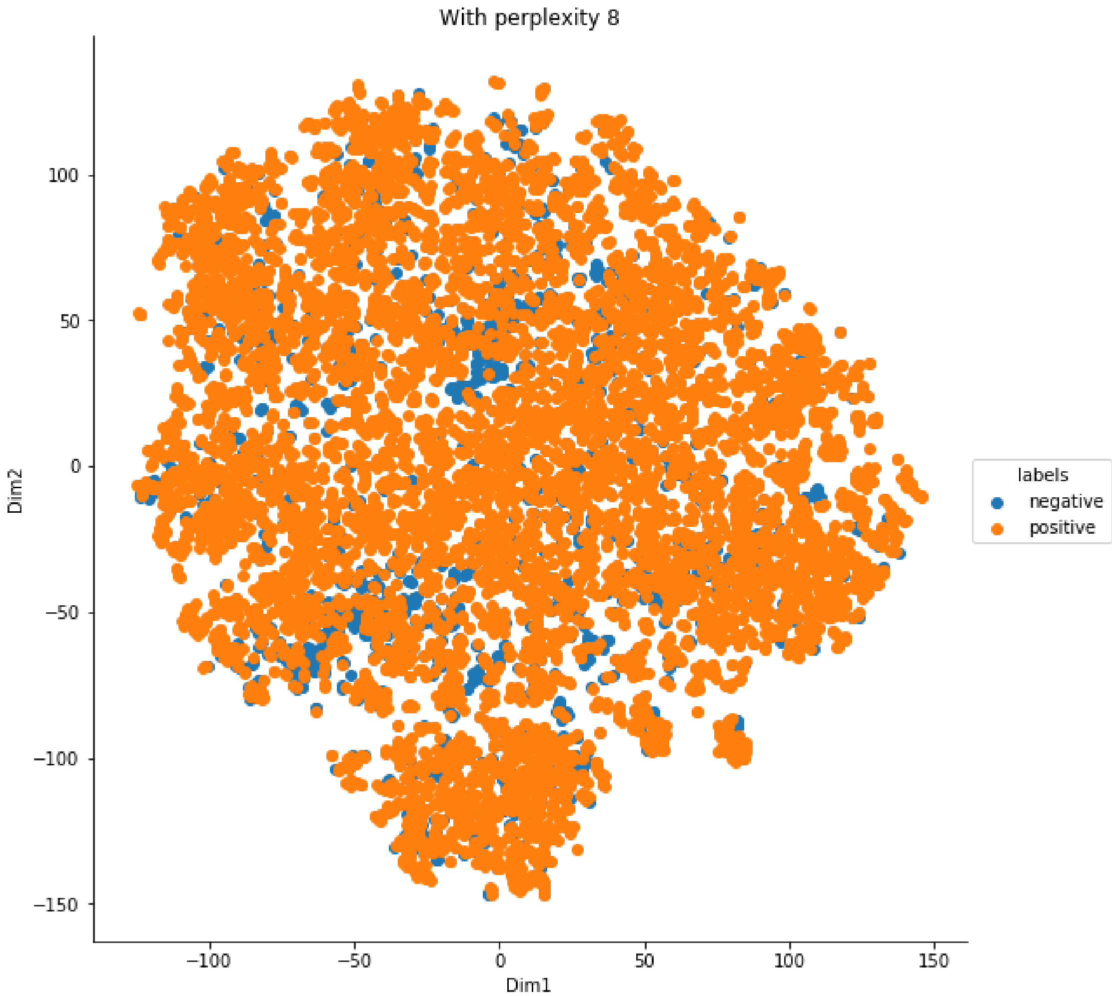
In [59]:
```
# Labelling data generated by tsne
labeled_tsne_data4 = np.vstack((tfidf_w2v_tsne_data4.T,data_labels)).T

# Creating dataframe
tfidf_w2v_df4 = pd.DataFrame(data=labeled_tsne_data4,columns=("Dim1","Dim2","labels"))

# Plotting the tsne plot with perplexity 50
sns.FacetGrid(tfidf_w2v_df4,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_le
plt.title("With perplexity 15")
plt.show()
```



In [60]:
```
# Plotting TSNE with perplexity 15
tsne_model5 = TSNE(n_components=2,n_iter=2500,perplexity=8,random_state=42)
tfidf_w2v_tsne_data5 = tsne_model5.fit_transform(avg_tfidf_w2v)
tfidf_w2v_tsne_data5.shape
```

Out[60]: (10000, 2)

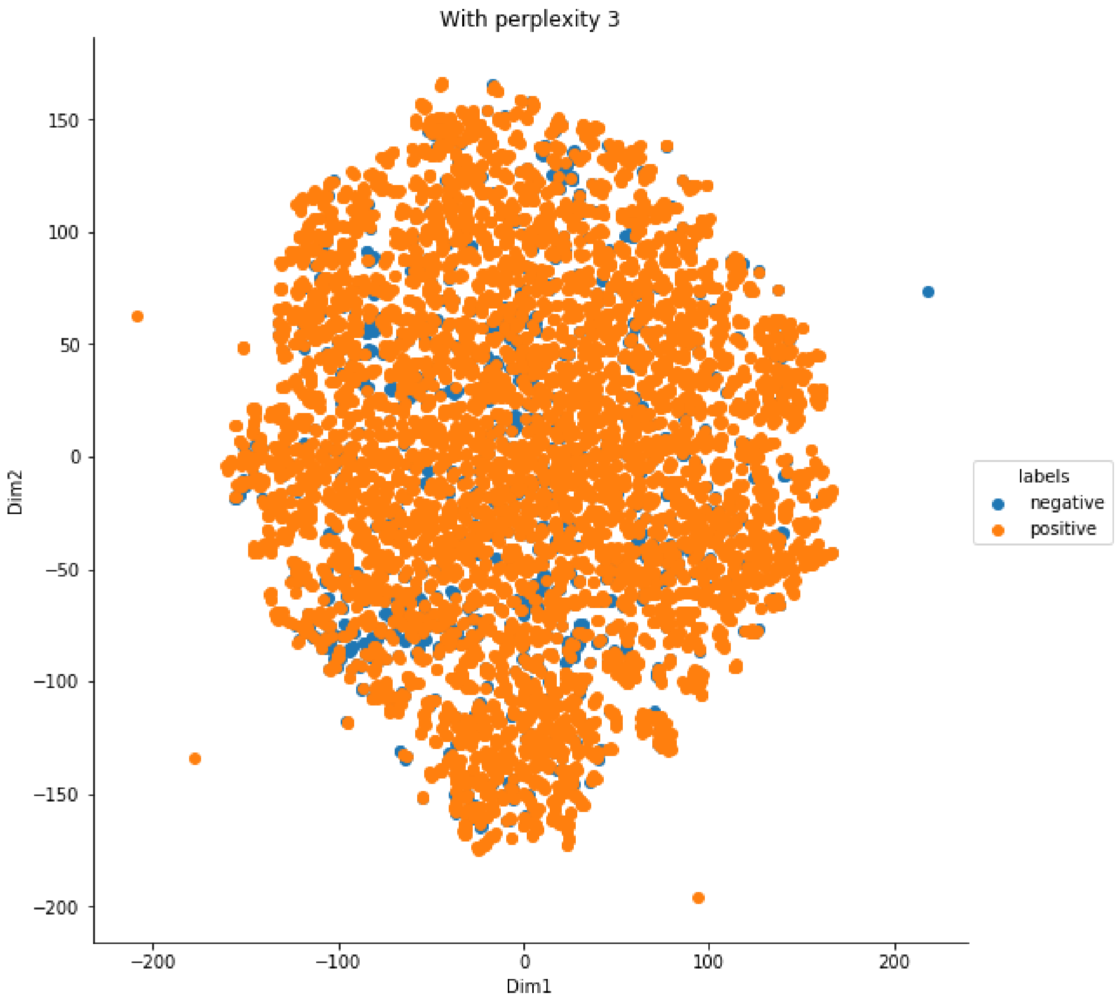
In [61]:
```
# Labelling data generated by tsne
labeled_tsne_data5 = np.vstack((tfidf_w2v_tsne_data5.T,data_labels)).T

# Creating dataframe
tfidf_w2v_df5 = pd.DataFrame(data=labeled_tsne_data5,columns=("Dim1","Dim2","labels")

# Plotting the tsne plot with perplexity 50
sns.FacetGrid(tfidf_w2v_df5,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_le
plt.title("With perplexity 8")
plt.show()
```



In [63]:
```
# Plotting TSNE with perplexity 3
tsne_model6 = TSNE(n_components=2,n_iter=2500,perplexity=3,random_state=42)
tfidf_w2v_tsne_data6 = tsne_model6.fit_transform(avg_tfidf_w2v)
tfidf_w2v_tsne_data6.shape
```

Out[63]: (10000, 2)

In [64]:
```
# Labelling data generated by tsne
labeled_tsne_data6 = np.vstack((tfidf_w2v_tsne_data6.T,data_labels)).T

# Creating dataframe
tfidf_w2v_df6 = pd.DataFrame(data=labeled_tsne_data6,columns=("Dim1","Dim2","labels")

# Plotting the tsne plot with perplexity 50
sns.FacetGrid(tfidf_w2v_df6,hue='labels',size=8).map(plt.scatter,"Dim1","Dim2").add_le
plt.title("With perplexity 3")
plt.show()
```



11 Avg word2vec for bigrams and trigrams

In [10]:
```
import nltk
from nltk import word_tokenize
```

```
from nltk import ngrams

bigram_text=[]

for sent in data["Cleaned_Text"].values:
    nltk_tokens = nltk.word_tokenize(sent)
    bigram_text.append(list(nltk.bigrams(nltk_tokens)))

print(bigram_text[1])
```

```
[('purchas', 'deal'), ('deal', 'offer'), ('offer', 'bought'), ('bought', 'still'), ('still', 's
```

```
In [11]: bigram_sent=[]
         for sent in bigram_text:
             for bi in sent:
                 bigram_sent.append(bi)
         print(bigram_sent[1])
```

```
('outpost', 'former')
```

```
In [12]: import gensim
         my_model2 = gensim.models.Word2Vec(bigram_sent,min_count=5,workers=4,size=50)
```

```
C:\Users\rites\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserWarning: detected Windows
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
In [13]: print(bigram_sent[10])
```

```
('weird', 'say')
```

```
In [14]: my_model2.wv.most_similar(bigram_sent[10])
```

```
C:\Users\rites\Anaconda3\lib\site-packages\gensim\matutils.py:737: FutureWarning: Conversion of
  if np.issubdtype(vec.dtype, np.int):
```

```
Out[14]: [('describ', 0.9451719522476196),
          ('stand', 0.944794774055481),
          ('admit', 0.9383567571640015),
          ('strang', 0.9377766847610474),
          ('okay', 0.9329333901405334),
          ('test', 0.9325597286224365),
          ('funni', 0.9310952425003052),
          ('matter', 0.9293650388717651),
          ('suppos', 0.9288748502731323),
          ('impress', 0.928593635559082)]
```

```
In [15]: print(my_model2.wv[bigram_sent[10]])
```

```
[[-0.51019716  0.13118035 -0.00433999 -0.09945925 -0.09744778  0.23318075
  -0.19988035  0.01058642  0.14830817 -0.22109908 -0.1173292   0.15437669
   0.18806228  0.04856871 -0.03935098  0.2713593   0.1712361  -0.02838839
   0.46311653 -0.30730173 -0.12078585 -0.04166156  0.31464648  0.3259654
   0.09099427 -0.01846796  0.19278207  0.08534543  0.13372786  0.2708484
  -0.16482872  0.10811605 -0.46252722 -0.2451201   0.26041391  0.40866703
   0.14957316  0.10797484  0.09925684  0.2988237  -0.40922514 -0.29694206
  -0.38575897  0.26434273 -0.45921993  0.02370872  0.03093297 -0.08074185
  -0.10076997 -0.39050767]
 [-0.45588237  0.30373088 -0.01060507 -0.4408102   0.26133028 -0.04174429
  -0.07877178  0.22656715  0.01096593  0.15648223 -0.42675918 -0.19082311
   0.31507146 -0.2118336  -0.4099184  -0.00087917  0.0076693   0.01832992
   0.3779047  -0.5559829  -0.15242742 -0.01281069  0.56367105  0.79541194
  -0.01355088 -0.07244869  0.07926509  0.07102835  0.3500923   0.19711877
  -0.39236158  0.0050509  -0.8508848  -0.39520693  0.37129703  0.382944
   0.62474525  0.13238692  0.14716162  0.2183286  -0.59827274 -0.20497093
  -0.23269568  0.45554897 -0.31883857 -0.12102558 -0.07508091  0.12208752
  -0.6436979  -0.49043328]]
```

```
In [16]: bigram_avg_w2v=[]
         for sent in bigram_text:
             i=0
             sum_avg_w2v = np.zeros((2,50),dtype=np.float32) # Matrix with 2 rows and 50 colum
             for bigrams in sent:
                 try:
                    i=i+1
                    sum_avg_w2v += my_model2.wv[bigrams]
                 except:
                     pass
             sum_avg_w2v /= i

             bigram_avg_w2v.append(sum_avg_w2v)

         print(len(bigram_avg_w2v))
```

```
10000
```

```
In [17]: bigram_w2v_single= []

         for i in range(len(bigram_avg_w2v)):
             bigram_w2v_single.append((bigram_avg_w2v[i][0]+bigram_avg_w2v[i][1])/2)

         print(bigram_w2v_single[1])
```

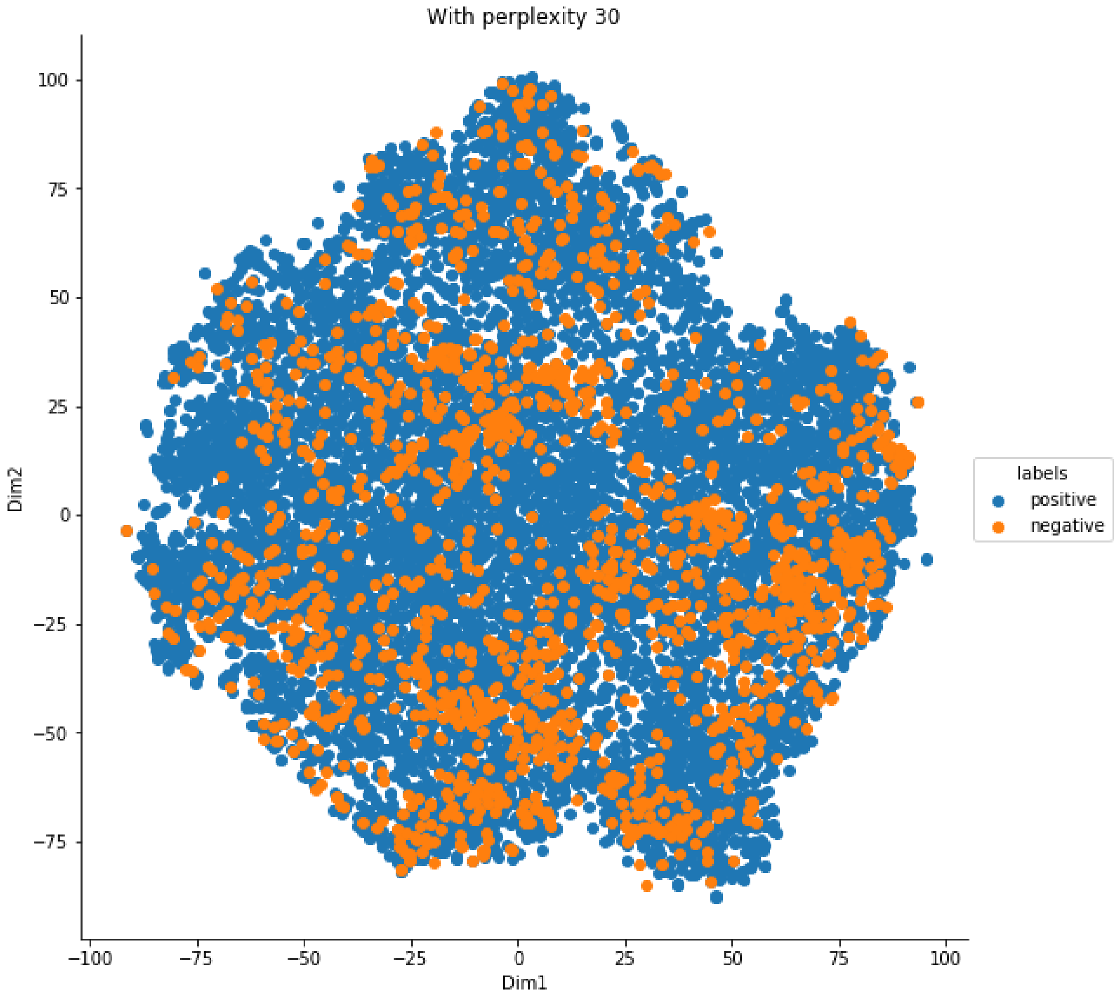
```
[-0.33190018 -0.00081101 -0.1533282  -0.17168698  0.21008298  0.13582641
 -0.21355072 -0.05824103 -0.11404213 -0.29445806 -0.26894093  0.26720852
 -0.0699385  -0.08382402 -0.04808334  0.42093933  0.3425404  -0.10051759
  0.34539962 -0.43804106 -0.14436923  0.05181272  0.4127637   0.49723566
 -0.05185822  0.00315836  0.30943936  0.11300044  0.09627882  0.19805157
 -0.5844423   0.05824289 -0.56512153 -0.11371472  0.26796752  0.42402297
  0.20898855  0.09070085  0.11938745  0.07340158 -0.44066793 -0.17271493
 -0.1274535   0.33922085 -0.4960425   0.12927803  0.00489633  0.14177653
 -0.24405642 -0.48890835]
```

In [18]:
```
# This replaces NAN with 0 and infinite number with large values
bigram_w2v_nan = np.nan_to_num(bigram_w2v_single,copy=True)
```

In [19]:
```
# Labels of 10k sampled points
data_labels = data["Score"]
```

In [38]:
```
from sklearn.manifold import TSNE

# Tsne model
tsne_bigram_w2v = TSNE(n_components=2,n_iter=3000,perplexity=30,random_state=42)

tsne_w2v_bigram = tsne_bigram_w2v.fit_transform(bigram_w2v_nan)

labeled_data = np.vstack((tsne_w2v_bigram.T,data_labels)).T

avg_bigram_df1 = pd.DataFrame(data=labeled_data,columns=("Dim1","Dim2","labels"))
```

In [23]:
```
sns.FacetGrid(avg_bigram_df1,hue="labels",size=8).map(plt.scatter,"Dim1","Dim2").add_l
plt.title("With perplexity 30")
plt.show()
```



11.0.1 For Trigrams

```
In [21]: trigrams_text=[]

         for sent in data["Cleaned_Text"].values:
             nltk_tokens = nltk.word_tokenize(sent)
             trigrams_text.append(list(nltk.trigrams(nltk_tokens)))

         print(trigrams_text[1])
```

```
[('purchas', 'deal', 'offer'), ('deal', 'offer', 'bought'), ('offer', 'bought', 'still'), ('bou
```

```
In [22]: trigrams_sent=[]
         for sent in trigrams_text:
             for tri in sent:
```

```
                trigrams_sent.append(tri)
        print(trigrams_sent[1])
```

```
('outpost', 'former', 'british')
```

```
In [25]: my_model3 = gensim.models.Word2Vec(trigrams_sent,min_count=1,workers=4,size=50)
```

```
In [30]: my_model2.wv.most_similar(trigrams_sent[10])
```

```
C:\Users\rites\Anaconda3\lib\site-packages\gensim\matutils.py:737: FutureWarning: Conversion of
  if np.issubdtype(vec.dtype, np.int):
```

```
Out[30]: [('servic', 0.9189201593399048),
          ('seller', 0.9183892011642456),
          ('via', 0.9117430448532104),
          ('sign', 0.9116343259811401),
          ('afford', 0.9083232879638672),
          ('thank', 0.9082056879997253),
          ('sell', 0.9080580472946167),
          ('deliveri', 0.901093065738678),
          ('check', 0.8987296223640442),
          ('search', 0.8987213969230652)]
```

```
In [31]: print(my_model2.wv[trigrams_sent[10]])
```

```
[[-0.31321147 -0.23278059 -0.12031627 -0.30655092  0.26879206  0.08715051
  -0.08980771  0.39039743 -0.14326167 -0.5291992  -0.6793888   0.26077244
  -0.4224834  -0.34045848 -0.1886941   0.29990932  0.3694962  -0.42752686
   0.2805618  -0.6556518  -0.27990696  0.36509198  0.8658075   1.0363115
  -0.30666173 -0.04450823  0.3032083   0.34514028  0.09466254  0.3456346
  -0.8562289  -0.0052566  -0.5547658  -0.01195244  0.31145528  0.2698274
   0.11548853  0.4776666  -0.04546773 -0.11346944 -0.07688263 -0.01839859
   0.41379294  0.39336884 -0.4560071  -0.04090242  0.20128472  0.20350628
  -0.1200944  -0.4655758 ]
 [-0.23966749 -0.08886002 -0.1421795  -0.14382182  0.525873   -0.09357084
  -0.32743806 -0.31808868 -0.3845229  -0.3884817  -0.6462514   0.26957
  -0.17260244  0.0237591  -0.10726806  0.6929937   0.52766675 -0.11351038
   0.3078039  -0.4726318  -0.1537959   0.33891562  0.43177855  0.4685934
  -0.00790565  0.2551028   0.513425    0.35015714  0.2864014   0.00904496
  -0.9548265   0.16519603 -0.7908012   0.15197091  0.05801275  0.7645827
   0.43045586 -0.407795    0.176224   -0.25003123 -0.713002   -0.00490412
   0.0687454   0.38158098 -0.29261205  0.33886802 -0.17599168  0.12542097
  -0.26204675 -0.7775817 ]
 [-0.3022045  -0.04375471 -0.22075593 -0.16194615  0.25011835 -0.01606613
  -0.2715094  -0.07867682 -0.19430606 -0.26077235 -0.6705177   0.2327599
  -0.29327163 -0.28714004  0.03994046  0.48950773  0.23063238 -0.21607883
   0.51734716 -0.38103595 -0.31710044  0.21176757  0.72096467  0.44120505
```

```
  -0.16376688  0.30975994  0.23597108  0.3168228   0.14424385  0.1544022
  -0.70182896  0.05320806 -0.6126367   0.18857488  0.26028118  0.5244142
   0.26754937  0.03551929 -0.03680459 -0.03203771 -0.34116033  0.05119008
   0.02349196  0.44799352 -0.31603658  0.10723455 -0.00152104  0.02847539
   0.04386184 -0.4296276 ]]
```

In [32]:
```python
trigram_avg_w2v=[]
for sent in trigrams_text:
    i=0
    sum_avg_w2v = np.zeros((3,50),dtype=np.float32) # Matrix with 2 rows and 50 colum
    for trigrams in sent:
        try:
           i=i+1
           sum_avg_w2v += my_model2.wv[trigrams]
        except:
            pass
    sum_avg_w2v /= i

    trigram_avg_w2v.append(sum_avg_w2v)

print(len(trigram_avg_w2v))
```

```
C:\Users\rites\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: RuntimeWarning: invalid va
  # This is added back by InteractiveShellApp.init_path()


10000
```

In [33]:
```python
trigram_w2v_single= []

for i in range(len(trigram_avg_w2v)):
    trigram_w2v_single.append((trigram_avg_w2v[i][0]+trigram_avg_w2v[i][1]+trigram_avg

print(trigram_w2v_single[1])
```

```
[-0.347075   -0.00640877 -0.15175535 -0.16976285  0.20396978  0.14358382
 -0.22862238 -0.06801428 -0.10387508 -0.27240914 -0.24357803  0.24291806
 -0.05596122 -0.07675391 -0.04641176  0.4133766   0.32756463 -0.10089793
  0.3646222  -0.4388411  -0.134185    0.02864283  0.3966653   0.47828677
 -0.06003954  0.00882236  0.30244717  0.10214297  0.10400809  0.20552266
 -0.5574692   0.04372364 -0.5329375  -0.134611    0.27534074  0.4150974
  0.20065145  0.09347624  0.101322    0.08862144 -0.44102526 -0.17951286
 -0.14554326  0.3331599  -0.49496162  0.13171978 -0.00268571  0.12470537
 -0.24015374 -0.4773384 ]
```

In [34]:
```python
trigram_w2v_nan = np.nan_to_num(trigram_w2v_single,copy=True)
```

In [36]:
```
# Training TSNE model for trigrams
tsne_trigram_w2v = TSNE(n_components=2,n_iter=3000,perplexity=30,random_state=42)

tsne_w2v_trigram = tsne_trigram_w2v.fit_transform(trigram_w2v_nan)

labeled_data1 = np.vstack((tsne_w2v_trigram.T,data_labels)).T

avg_trigram_df1 = pd.DataFrame(data=labeled_data1,columns=("Dim1","Dim2","labels"))
```

In [37]:
```
sns.FacetGrid(avg_trigram_df1,hue="labels",size=8).map(plt.scatter,"Dim1","Dim2").add_
plt.title("With perplexity 30")
plt.show()
```