In [1]:

```
import ipyparallel as ipp
from ipyparallel.joblib import IPythonParallelBackend
from sklearn.externals.joblib import Parallel, parallel_backend, regist
er_parallel_backend
from scipy.sparse import dok_matrix, lil_matrix, coo_matrix
import re
import sqlite3
import math
import gensim
import datetime
from gensim.models import Word2Vec
import pandas as pd
import numpy as np
import warnings
import nltk
import matplotlib.pyplot as plt
from nltk.stem import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn import cross_validation

c = ipp.Client()
bview = c.load_balanced_view()
register_parallel_backend('ipyparallel', lambda : IPythonParallelBacken
d(view=bview))
```

```
warnings.filterwarnings("ignore")
con = sqlite3.connect('database.sqlite')
data = pd.read_sql_query("""SELECT * FROM Reviews""",con)

data = data[data.Score != 3]
def polarity(value):
    if value>3:
        return "positive"
    else:
        return "negative"
temp_score = data['Score']
temp_polarity = temp_score.map(polarity)
data['Score'] = temp_polarity

data = data.sort_values('ProductId', axis = 0, ascending = True)
data = data.drop_duplicates(subset={"UserId","ProfileName","Time","Tex
t"},keep='first',inplace= False)

data = data[data.HelpfulnessNumerator<=data.HelpfulnessDenominator]

#converting UNIX epoch time to python datetime object
tim_list = []
for tim in data['Time'].values:
    value = datetime.datetime.utcfromtimestamp(tim)
    s = value.strftime('%Y-%m-%d %H:%M:%S')
    tim_list.append(s)
data['Time'] = tim_list
```

```
C:\Users\lenovo\Anaconda3\lib\site-packages\gensim\utils.py:1209: UserW
arning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
C:\Users\lenovo\Anaconda3\lib\site-packages\sklearn\cross_validation.p
y:41: DeprecationWarning: This module was deprecated in version 0.18 in
favor of the model_selection module into which all the refactored class
es and functions are moved. Also note that the interface of the new CV
iterators are different from that of this module. This module will be r
emoved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [48]:
```
#taking top2000 positive rows and top2000 negative
pos = data[data.Score == 'positive']
pos_2000 = pos[:1000]
neg = data[data.Score == 'negative']
neg_2000 = neg[:1000]

#4k review dataset; 2000 positive and 2000 negative
datanew = pos_2000.append(neg_2000)
datanew= datanew.sort_values('Time', axis = 0, ascending = True)
polarity = datanew.Score[:2000].values
```

In [49]:
```
# pre processing
cleaned_final_data = []
list_patterns = [r'\d+','<.*?>','[^a-zA-Z]',' +',r'^https?:\/\/.*[\r\n]
*',r'\b\w{1,3}\b']    # list of patterns
reg_expression = [re.compile(p) for p in list_patterns]

stop_words = stopwords.words('english')

sno = SnowballStemmer('english')
with parallel_backend('ipyparallel'):
    for review in datanew['Text'].values:
        clean_review = re.sub(reg_expression[4],' ',review) # remove ur
l
        clean_review = re.sub(reg_expression[1],' ',clean_review)     #
 remove html tags
        clean_review = re.sub(reg_expression[0],'',clean_review)  # rem
ove sequence of digit
        clean_review = re.sub(reg_expression[2],' ',clean_review) # rem
ove everything except alphabets.
        clean_review = re.sub(reg_expression[5],'',clean_review)     #
 remove words less than three characters
        clean_review = re.sub(reg_expression[3],' ',clean_review) # rep
lace two consecutive space into one
        temp_review = clean_review.lower()                          # all
 reviews to lowercase
        temp_review = ' '.join([word for word in temp_review.split()\
                        if word not in stop_words])      #removing
 stopwords
```

```
        temp_review = ' '.join([sno.stem(word) for word in temp_review.
split()])  #Snowball stemming

        cleaned_final_data.append((temp_review))

#final data after cleaning
datanew['Text'] = cleaned_final_data
```

In [50]:
```
#BOW
with parallel_backend('ipyparallel'):
    vect_pos = CountVectorizer()
    matrix_BOW_POS = vect_pos.fit_transform(datanew['Text'].values)

x_train, x_test,polarity_train, polarity_test = train_test_split(matrix
_BOW_POS,polarity,test_size=0.3,random_state=0)
```

In [60]:
```
print("shape of x_train is = ",x_train.shape)
print("shape of polarity_train is = ",polarity_train.shape)
all_score_acc_mean = []
with parallel_backend('ipyparallel'):
    for i in range(1,30):
        knn = KNeighborsClassifier(n_neighbors=i)
        scores = cross_val_score(knn,x_train, polarity_train, cv=10, sc
oring='accuracy')

        all_score_acc_mean.append(scores.mean())
error_all = []
for i in all_score_acc_mean:
    error_all.append(1-i)
```

```
shape of x_train is =  (1400, 6938)
shape of polarity_train is =  (1400,)
```

```
---------------------------------------------------------------------
----
IndexError                                Traceback (most recent call l
ast)
<ipython-input-60-5a90689bf80d> in <module>()
      5     for i in range(1,30):
```

```
      6         knn = KNeighborsClassifier(n_neighbors=i)
----> 7         scores = cross_val_score(knn,x_train, polarity_train.re
shape(-1,1), cv=3, scoring='accuracy')
      8
      9         all_score_acc_mean.append(scores.mean())

~\Anaconda3\lib\site-packages\sklearn\cross_validation.py in cross_val_
score(estimator, X, y, scoring, cv, n_jobs, verbose, fit_params, pre_di
spatch)
   1570     X, y = indexable(X, y)
   1571
-> 1572     cv = check_cv(cv, X, y, classifier=is_classifier(estimator)
)
   1573     scorer = check_scoring(estimator, scoring=scoring)
   1574     # We clone the estimator to make sure that all the folds ar
e

~\Anaconda3\lib\site-packages\sklearn\cross_validation.py in check_cv(c
v, X, y, classifier)
   1833         if classifier:
   1834             if type_of_target(y) in ['binary', 'multiclass']:
-> 1835                 cv = StratifiedKFold(y, cv)
   1836             else:
   1837                 cv = KFold(_num_samples(y), cv)

~\Anaconda3\lib\site-packages\sklearn\cross_validation.py in __init__(s
elf, y, n_folds, shuffle, random_state)
    568         for test_fold_idx, per_label_splits in enumerate(zip(*p
er_label_cvs)):
    569             for label, (_, test_split) in zip(unique_labels, pe
r_label_splits):
--> 570                 label_test_folds = test_folds[y == label]
    571                 # the test split can be too big because we used
    572                 # KFold(max(c, self.n_folds), self.n_folds) ins
tead of

IndexError: too many indices for array
```

In [ ]:
```
optimal_k = error_all.index(min(error_all))
```

```
print("BOW optimal k = ",optimal_k)
with parallel_backend('ipyparallel'):
    knn_optimal_k = KNeighborsClassifier(n_neighbors=optimal_k,algorith
m='kd_tree')
    knn_optimal_k.fit(x_train,polarity_train)
    predic = knn_optimal_k.predict(x_test)
    acc = accuracy_score(polarity_test,predic) * float(100)
print('BOW Generalised accuracy for optimal k is %d%%' % acc)
print("\n\n\n")
```

In [ ]:
```
#TF-iDF
with parallel_backend('ipyparallel'):
    tfidf_vect = TfidfVectorizer()
    tfidf_data_vect = tfidf_vect.fit_transform(datanew['Text'].values)

    x_train, x_test, polarity_train, polarity_test = train_test_split(t
fidf_data_vect.toarray(),polarity.values,test_size=0.3,random_state=0)

    for i in range(1,30):
        knn = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
        scores = cross_val_score(knn, x_train, polarity_train, cv=10, s
coring='accuracy')
        all_score_acc_mean.append(scores.mean())

error_all = []
for i in all_score_acc_mean:
    error_all.append(1-i)
```

In [ ]:
```
optimal_k = error_all.index(min(error_all))
print("TFIDF optimal k = ",optimal_k)
with parallel_backend('ipyparallel'):
    knn_optimal_k = KNeighborsClassifier(n_neighbors=optimal_k,algorith
m='kd_tree')
    knn_optimal_k.fit(x_train,polarity_train)
    predic = knn_optimal_k.predict(x_test)
    acc = accuracy_score(polarity_test,predic) * float(100)
print('TFIDF Generalised accuracy for optimal k is %d%%' % acc)
print("\n\n\n")
```

In [ ]:
```
#AVG Word2Vec
data_words = []
temp = []
for review in datanew['Text'].values:
    temp = (review).split(' ')
    data_words.append(temp)

dimension_size_for_each_word = 50
with parallel_backend('ipyparallel'):
    my_w2v_model = Word2Vec(data_words,min_count = 5,workers=1,size=dim
ension_size_for_each_word)
    vocab_word_list = list(my_w2v_model.wv.vocab)
    sum_of_all_w2v = np.zeros((dimension_size_for_each_word),np.float64
)
```

In [ ]:
```
list_review_avgw2v = []
with parallel_backend('ipyparallel'):
    for review in datanew['Text'].values:
        j = 0;
        while j < len(vocab_word_list):
            if vocab_word_list[j] in review.split(' '):
                sum_of_all_w2v = sum_of_all_w2v + my_w2v_model[vocab_wo
rd_list[j]]
            j += 1;
        list_review_avgw2v.append((sum_of_all_w2v/len(review.split(' '
))))
```

In [ ]:
```
x_train, x_test, polarity_train, polarity_test = train_test_split(list_
review_avgw2v, polarity.values,test_size=0.3,random_state=0)

for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
    scores = cross_val_score(knn,x_train, polarity_train, cv=10, scorin
g='accuracy')
    all_score_acc_mean.append(scores.mean())

error_all = []
```

```
for i in all_score_acc_mean:
    error_all.append(1-i)
```

In [ ]:
```
optimal_k = error_all.index(min(error_all))
print("AVG-W2V optimal k = ",optimal_k)
with parallel_backend('ipyparallel'):
    knn_optimal_k = KNeighborsClassifier(n_neighbors=optimal_k,algorith
m='kd_tree')
    knn_optimal_k.fit(x_train,polarity_train)
    predic = knn_optimal_k.predict(x_test)
    acc = accuracy_score(polarity_test,predic) * float(100)
print('AVG-W2V Generalised accuracy for optimal k is %d%%' % acc)
print("\n\n\n")
```

In [ ]:
```
#TFIDF W2V
ar = tfidf_data_vect.toarray()
tfidf_w2v_list = []
rev_index = 0;
with parallel_backend('ipyparallel'):
    for review in datanew['Text'].values:
        j=0;
        w2v = np.zeros((dimension_size_for_each_word),np.float64);
        sum_tfidf = 0;
        while j < len(vocab_word_list):
            if vocab_word_list[j] in  review.split(' '):
                val = tfidf_vect.vocabulary_.get(vocab_word_list[j])
                tf = (ar[rev_index,val])
                if np.isscalar(tf):
                    w2v = w2v + (np.dot(tf,my_w2v_model[vocab_word_list
[j]]))
                    sum_tfidf = sum_tfidf + tf
            j += 1;
        w = (w2v/sum_tfidf)
        if(np.isnan(w).any()):
            polarity = polarity.drop(polarity.index[j])
        else:
            tfidf_w2v_list.append(w)
        rev_index += 1;
```

In [ ]:
```
x_train, x_test, polarity_train, polarity_test = train_test_split(tfidf
_w2v_list,polarity.values,test_size=0.3,random_state=0)
with parallel_backend('ipyparallel'):
    for i in range(1,30):
        knn = KNeighborsClassifier(n_neighbors=i,algorithm='kd_tree')
        scores = cross_val_score(knn,x_train, polarity_train, cv=10, sc
oring='accuracy')
        all_score_acc_mean.append(scores.mean())

error_all = []
for i in all_score_acc_mean:
    error_all.append(1-i)
```

In [ ]:
```
optimal_k = error_all.index(min(error_all))
print("TFIDF-W2V optimal k = ",optimal_k)
with parallel_backend('ipyparallel'):
    knn_optimal_k = KNeighborsClassifier(n_neighbors=optimal_k,algorith
m='kd_tree')
    knn_optimal_k.fit(x_train,polarity_train)
    predic = knn_optimal_k.predict(x_test)
    acc = accuracy_score(polarity_test,predic) * float(100)
print('TFIDF-W2V Generalised accuracy for optimal k is %d%%' % acc)
```