

Information about data:

->We have the amazon reviews dataset from kaggle

->Reviews are given for the product

->The features of the data were:

Id

ProductId- unique identifier for the product

UserId- unique identifier for the user

ProfileName

HelpfulnessNumerator- number of users who found the review helpful

HelpfulnessDenominator- number of users who indicated whether they found the review helpful or not

Score-rating between 1 and 5

Time-timestamp for the review

Summary- brief summary of the review

Text- text of the review

-> Based on the score of the review we classify them into positive and negative

Number of reviews: 568,454



Objective:

-> Cleaning the dataset by classifying them into positive and negative reviews based on the rating provided and removing the duplicates

-> Converting the text data to vectors by using Bag of words, Tfidf, word2vec, Average word2vec

-> Implementing Support Vector Machines with kernel as RBF

-> Implementing Linear Kernel SVM

-> Implementing nu-SVM

-> Using Grid Search and Random Search to determine the best hyper parameters

-> In RBF-SVM the hyper parameters are "C" and "GAMMA"

-> To check the performance with different "C" and "GAMMA" values

Importing the required libraries to process the data

In [1]:

```
from sklearn.feature_extraction.text import
TfidfTransformer, TfidfVectorizer, CountVectorizer
import sqlite3
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

-> Establishing the connection and loading the data

-> Shape of the data

-> Dimensionality of the data

-> Attributes of the data

-> Sample of data

In [2]:

```
connection = sqlite3.connect("database.sqlite")
data = pd.read_sql_query("SELECT * FROM Reviews WHERE Score != 3", connection)
print(data.shape)
print(data.ndim)
print(data.columns)
print(data.head(5))
```

(525814, 10)

2

Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
 'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
 dtype='object')

	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...



Data Pre-processing:

-> Cleaning the data

-> Removing duplicates

In [3]:

```
def score(n):
    if n>3:
        return 'positive'
    return 'negative'
rating = data['Score']
rating = rating.map(score)
data['Score'] = rating
data.head(6)
```

Out [3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulness
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0

2	3	ProductId	ABXLMWJIXXAN	UserId	Natalia Corres "Natalia Corres"	HelpfulnessNumerator	Helpfulness
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	
5	6	B006K2ZZ7K	ADT0SRK1MGOEU	Twoapennything	0	0	



In [4]:

```
sorting_data = data.sort_values('ProductId',axis=0,ascending=True,inplace=False,kind='quicksort',na_position='last')
```

In [5]:

```
cleaned_data = sorting_data.drop_duplicates(subset={"UserId","ProfileName",  
"Time","Text"},keep='first',inplace=False)
```

Information about the cleaned data:

- > Shape of the data
- > Dimensionality of the data
- > Attributes of the data
- > Sample of cleaned data

In [6]:

```
print(cleaned_data.shape)  
print(cleaned_data.ndim)  
print(cleaned_data.columns)  
print(cleaned_data.head(5))
```

(364173, 10)

2

```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',  
      'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],  
      dtype='object')
```

	Id	ProductId	UserId	ProfileName	\
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	
138688	150506	0006641040	A2IW4PEEK02R0U	Tracy	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg "(Kate)"	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
138706	0	0	positive	939340800
138688	1	1	positive	1194739200
138689	1	1	positive	1191456000
138690	1	1	positive	1076025600
138691	3	4	positive	1018396800

	Summary	\
138706	EVERY book is educational	
138688	Love the book, miss the hard cover version	
138689	chicken soup with rice months	
138690	a good swingy rhythm for reading aloud	
138691	A great way to learn the months	

	Text
138706	this witty little book makes my son laugh at l...
138688	I grew up reading these Sendak books, and watc...
138689	This is a fun way for children to learn their ...
138690	This is a great little book to read aloud- it ...
138691	This is a book of poetry about the months of t...

Keeping in mind about the performance capability of the box and time efficiency here i am taking the subset of the data

In [7]:

```
sample_data = cleaned_data.sample(n=100000)
```

Information about the sampled data:

- > Shape of the data
- > Dimensionality of the data
- > Attributes of the data
- > Sample of modified data

In [8]:

```
print(sample_data.shape)  
print(sample_data.ndim)  
print(sample_data.columns)  
print(sample_data.head(5))
```

```
(100000, 10)
2
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')

```

	Id	ProductId	UserId	ProfileName	\
255735	277245	B000VK8AVK	A1HLYNGUBXXW9S	Maureen in WA	"maureeng"
173730	188447	B0051ZCRGY	A3L6O1NV34AJ2J	Richard Odat	"Slingboxer"
282621	306193	B000JSM344	A3E5D99WX49BK7		Michelle Cohn
16096	17583	B0000GH6UQ	A18ZDBQ8LUNVO6		Jessica McCormick
477841	516720	B007WTQAKQ	A2YAAF9BCQ4AJC		Leigh Somebody

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
255735	16	17	positive	1215561600
173730	0	0	positive	1345334400
282621	3	5	positive	1199750400
16096	0	0	positive	1318377600
477841	0	0	positive	1337904000

	Summary	\
255735	Excellent Chips!	
173730	Our Cocker LOVES Pegetables	
282621	Thank you so much!	
16096	So long, Swiss Miss!	
477841	Surprisingly Works!	

	Text
255735	These chips are really, really good. I don't ...
173730	Our Cocker LOVES Pegetables! She seems to rea...
282621	I use to be able to buy this in town and when ...
16096	A couple years ago, my husband pointed this co...
477841	Just on a whim, I thought I'd try this and it ...

SVM WITH RBF KERNEL:

BAG OF WORDS:

TIME BASED SPLITTING OF DATA:

-> Implementing Support Vector Classifier with "RBF" as kernel

-> Using both Grid Search and Random Search to determine the best hyper parameters

-> To check performance measure with different hyper parameter values

In [9]:

```
cv_data = sample_data.sort_values("Time",axis=0,ascending=True,kind='quicksort',na_position='last',inplace=False)
```

In [10]:

```
count_vectorizer = CountVectorizer()
cv_data_vect = count_vectorizer.fit_transform(cv_data['Text'].values)
```

In [11]:

```
score = sample_data['Score']
```

In [12]:

```
cv_data_vect.shape
```

Out[12]:

```
(100000, 61485)
```

In [13]:

```
xtrain = cv_data_vect[0:70000]
xtest  = cv_data_vect[70000:]
ytrain = score[0:70000]
ytest  = score[70000:]
```

In [14]:

```
print(xtrain.shape)
print(xtest.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(70000, 61485)
(30000, 61485)
(70000,)
(30000,)
```

BAG OF WORDS: GRID SEARCH TO FIND HYPER PARAMETERS

In [15]:

```
parameters = {'kernel':['rbf'], 'C':[0.01,0.1,1], 'gamma':[0.01,0.1,1]}
```

In [16]:

```
classifier = SVC()
```

In [17]:

```
model = GridSearchCV(classifier,param_grid=parameters,scoring='accuracy')
model.fit(xtrain,ytrain)
```

Out[17]:

```
GridSearchCV(cv=None, error_score='raise',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
                           max_iter=-1, probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             fit_params=None, iid=True, n_jobs=1,
             param_grid={'kernel': ['rbf'], 'C': [0.01, 0.1, 1], 'gamma': [0.01,
0.1, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='accuracy', verbose=0)
```

In [26]:

```
classif = SVC(C=1,gamma=1,kernel='rbf')
classif.fit(xtrain,ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest,pred)
print(acc)
```

0.8419

In [27]:

```
classif = SVC(C=0.1,gamma=1,kernel='rbf')
classif.fit(xtrain,ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest,pred)
print(acc)
```

0.8419

In [28]:

```
classif = SVC(C=0.1,gamma=0.1,kernel='rbf')
classif.fit(xtrain,ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest,pred)
print(acc)
```

0.8419

In [29]:

```
classif = SVC(C=0.1,gamma=0.01,kernel='rbf')
classif.fit(xtrain,ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest,pred)
print(acc)
```

0.8419

In [30]:

```
classif = SVC(C=0.01,gamma=1,kernel='rbf')
classif.fit(xtrain,ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest,pred)
print(acc)
```

0.8419

BAG OF WORDS: RANDOM SEARCH TO FIND HYPER PARAMETERS

In [15]:

```
import scipy
```

In [16]:

```
parameters = {'C': scipy.stats.randint.rvs(1,5,size=5), 'gamma':
scipy.stats.randint.rvs(1,5,size=5), 'kernel': ['rbf']}
```


In [17]:

```
from sklearn.linear_model import SGDClassifier
```

In [20]:

```
classifier = SVC()
```

In [24]:

```
model = RandomizedSearchCV(classifier,param_distributions=parameters,scoring='accuracy',cv=5,n_iter=5,n_jobs=-1)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
SVC(C=3, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=2, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [25]:

```
model
```

Out[25]:

```
RandomizedSearchCV(cv=5, error_score='raise',
                   estimator=SVC(C=1.0, cache_size=200, class_weight=None,
                                coef0=0.0,
                                decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
                                max_iter=-1, probability=False, random_state=None, shrinking=True,
                                tol=0.001, verbose=False),
                   fit_params=None, iid=True, n_iter=5, n_jobs=-1,
                   param_distributions={'C': array([2, 2, 3, 3, 4]), 'gamma':
array([3, 4, 3, 2, 4]), 'kernel': ['rbf']}},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score='warn', scoring='accuracy', verbose=0)
```

In [26]:

```
model.best_estimator_
```

Out[26]:

```
SVC(C=3, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=2, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [27]:

```
classif = SVC(C=3,gamma=2,kernel='rbf')
classif.fit(xtrain,ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest,pred)
print(acc)
```

```
0.8424666666666667
```

In [28]:

```
classif = SVC(C=2,gamma=2,kernel='rbf')
```

```
classif = SVC(C=1, gamma=2, kernel='rbf')
classif.fit(xtrain, ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest, pred)
print(acc)
```

0.8424666666666667

In [29]:

```
classif = SVC(C=1, gamma=2, kernel='rbf')
classif.fit(xtrain, ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest, pred)
print(acc)
```

0.8424666666666667

In [30]:

```
classif = SVC(C=3, gamma=1, kernel='rbf')
classif.fit(xtrain, ytrain)
pred = classif.predict(xtest)
acc = accuracy_score(ytest, pred)
print(acc)
```

0.8424666666666667

Observation:

-> BAG OF WORDS:

-> By using Grid Search with "RBF" as kernel the best values of hyper parameters were C = 1 and gamma = 3

-> By using Random search with "RBF" as kernel the best values of hyper parameters were C = 3 and gamma = 2

-> The accuracy by grid search is 84.19

-> The accuracy of random search is 84.24

SVM WITH LINEAR KERNEL:

TFIDF:

TIME BASED SPLITTING OF DATA:

-> Implementing Support Vector Classifier with Linear kernel

-> Using both Grid Search and Random Search to determine the best hyper parameters

-> To check performance measure with different hyper parameter values

In [31]:

```
tfidf = TfidfVectorizer(ngram_range=(1,2))
```

In [32]:

```
tfidf_data = sample_data.sort_values("Time",axis=0,ascending=True,kind='quicksort',na_position='last',inplace=False)
```

In [33]:

```
tfidf_vect_data = tfidf.fit_transform(tfidf_data['Text'].values)
```

In [34]:

```
tfidf_vect_data.shape
```

Out[34]:

```
(100000, 1283627)
```

In [36]:

```
sample = tfidf_vect_data[0:10000]  
score = tfidf_data['Score'][0:10000]
```

In [37]:

```
xtrain = sample[0:7000]  
xtest = sample[7000:]  
ytrain = score[0:7000]  
ytest = score[7000:]
```

In [38]:

```
print(xtrain.shape)  
print(ytrain.shape)  
print(xtest.shape)  
print(ytest.shape)
```

```
(7000, 1283627)  
(7000,)  
(3000, 1283627)  
(3000,)
```

TFIDF: GRID SEARCH TO FIND HYPER PARAMETERS

In [54]:

```
classifier = SGDClassifier(loss="hinge")
```

In [55]:

```
parameters = {'alpha':[0.01,0.1,1,10,100]}
```

In [57]:

```
model = GridSearchCV(classifier,param_grid=parameters,scoring='accuracy',cv=5)  
model.fit(xtrain,ytrain)
```

```
pred = model.predict(xtest)
score = accuracy_score(ytest, pred)
print(model)
print(score)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
```

```
"and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
```

```

sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)

```

```

    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)

```

```

GridSearchCV(cv=5, error_score='raise',
             estimator=SGDClassifier(alpha=0.0001, average=False,
class_weight=None, epsilon=0.1,
             eta0=0.0, fit_intercept=True, l1_ratio=0.15,
             learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
             n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
             shuffle=True, tol=None, verbose=0, warm_start=False),
             fit_params=None, iid=True, n_jobs=1,
             param_grid={'alpha': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='accuracy', verbose=0)
0.8936666666666667

```

```

/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/stochastic_gradient.py:128: FutureWarning: ma
x_iter and tol parameters have been added in <class
'sklearn.linear_model.stochastic_gradient.SGDClassifier'> in 0.19. If both
are left unset, they default to max_iter=5 and tol=None. If tol is not None
, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1
000, and default tol will be 1e-3.
    "and default tol will be 1e-3." % type(self), FutureWarning)

```

In [69]:

```
from sklearn.svm import LinearSVC
```

In [71]:

```

model = LinearSVC(C = 0.0001)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
print(score)

```

```
LinearSVC(C=0.0001, class_weight=None, dual=True, fit_intercept=True,
```



```
LinearSVC(C=0.001, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
0.8936666666666667
```

In [72]:

```
model = SVC(C = 0.01)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
print(score)
```

```
SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
     decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
     max_iter=-1, probability=False, random_state=None, shrinking=True,
     tol=0.001, verbose=False)
0.8936666666666667
```

TFIDF: RANDOM SEARCH TO FIND HYPER PARAMETERS

In [74]:

```
parameters = {'C': scipy.stats.randint.rvs(1,10,size=5)}
parameters
```

Out[74]:

```
{'C': array([8, 1, 9, 4, 5])}
```

In [75]:

```
classifier = LinearSVC(loss='hinge')
model = RandomizedSearchCV(classifier,param_distributions=parameters,scoring='accuracy',cv=5,n_iter=5,n_jobs=-1)
model.fit(xtrain,ytrain)
print(model.best_estimator_)
```

```
LinearSVC(C=8, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='hinge', max_iter=1000, multi_class='ovr',
          penalty='l2', random_state=None, tol=0.0001, verbose=0)
```

In [78]:

```
model = SVC(C = 8)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
```

```
SVC(C=8, cache_size=200, class_weight=None, coef0=0.0,
     decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
     max_iter=-1, probability=False, random_state=None, shrinking=True,
     tol=0.001, verbose=False)
```

In [81]:

```
print(score)
```

Out[81]:

87.14

In [79]:

```
model = SVC(C = 0.0001, gamma= 0.01)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
```

```
SVC(C=0.0001, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [82]:

```
print(score)
```

Out[82]:

87.14

Observation:

TFIDF:

-> By using Grid Search with Linear SVM the best values of hyper parameters were C = 0.001

-> By using Random search with Linear SVM the best values of hyper parameters were C = 8

-> The accuracy by grid search is 89.36

-> The accuracy of random search is 87.14

WORD2VEC:

CONSTRUCTING VECTOR REPRESENTATION OF EACH IN THE DATA BY USING WORD2VEC

In [83]:

```
import gensim
from gensim.models import word2vec
```

-> Importing the required libraries

-> Functions to clean the sentences

In [84]:

```
import re
def cleanhtml(sentence):
```

```

clean = re.compile("<.*?>")
cleantext = re.sub(clean, " ", sentence)
return cleantext
def cleanpunct(sentence):
    cleanr = re.sub(r"[?!|\\|'|#|.|,|)|(|/]", r' ', sentence)
    return cleanr

```

In [85]:

```

sorted_w2vec = sample_data.sort_values("Time", axis=0, ascending=True, kind='quicksort', na_position='last', inplace=False)

```

Information about the sorted data:

- > Shape of the data
- > Dimensionality of the data
- > Attributes of the data
- > Sample of modified data

In [86]:

```

print(sorted_w2vec.shape)
print(sorted_w2vec.ndim)
print(sorted_w2vec.columns)
print(sorted_w2vec.head(5))

```

(100000, 10)

2

```

Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')

```

	Id	ProductId	UserId	ProfileName
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera
417883	451903	B00004CXX9	A2DEE7F9XKP3ZR	jerome
1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie
346115	374421	B00004CI84	A1FJOY14X3MUHE	Justin Howard

	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
138683	2	2	positive	940809600
417838	0	0	positive	946857600
417883	0	1	positive	959990400
1146	7	7	positive	961718400
346115	2	2	positive	966297600

	Summary
138683	This whole series is great way to spend time w...
417838	FANTASTIC!
417883	Research
1146	Great Product
346115	A fresh, original film from master storyteller...

	Text
138683	I can remember seeing the show when it aired o...
417838	Beetlejuice is an excellent and funny movie. K...

```
417883 I'm getting crazy.<p>Is it really impossible t...
1146 This was a really good idea and the final prod...
346115 This is such a great film, I don't even know h...
```

In [87]:

```
i=0
sentences_list=[]
for sent in sorted_w2vec['Text'].values:
    filtered_sentences = []
    sent = cleanhtml(sent)
    for w in sent.split():
        for cleanedwords in cleanpunct(w).split():
            if(cleanedwords.isalpha()):
                filtered_sentences.append(cleanedwords.lower())
    sentences_list.append(filtered_sentences)
```

In [88]:

```
print(len(sentences_list))
print(type(sentences_list))
```

```
100000
<class 'list'>
```

In [89]:

```
w2vmodel =
gensim.models.Word2Vec(sentences_list,min_count=4,size=200,workers=4)
```

-> Most similar word

-> Similarity between the words

-> Dimensionality representation of a word

In [91]:

```
print(w2vmodel.most_similar("the"))
print(w2vmodel.similarity("this",'these'))
print(w2vmodel.wv['hello'])
```

```
[('foil', 0.3661514222621918), ('printed', 0.3577941656112671), ('marked',
0.3473374545574188), ('metal', 0.34421518445014954), ('contents', 0.3353686
3327026367), ('loosely', 0.33394986391067505), ('divided',
0.3335522413253784), ('each', 0.3309260606765747), ('torn',
0.3271692097187042), ('this', 0.326387882232666)]
0.15672317384225404
[ 7.46376738e-02 -3.91671211e-02 -9.74655803e-03  1.17327280e-01
 4.30680700e-02  1.42935008e-01  1.69160645e-02 -9.99073833e-02
-8.94245226e-03  7.31556639e-02  2.71437708e-02  8.49297866e-02
-3.79108302e-02 -4.03551161e-02  1.79700971e-01 -8.83629546e-02
 6.21749647e-02  1.70396626e-01 -9.88507271e-02 -3.23529937e-03
-1.54427374e-02 -3.88860442e-02  2.32633371e-02 -8.95061940e-02
-5.09793237e-02 -8.53051022e-02 -2.16724798e-01 -8.22060779e-02
 1.60596505e-01  2.38274857e-01  6.77032471e-02  2.92483121e-02
-1.52913973e-01 -8.54681283e-02  3.81469503e-02  1.63954377e-01
-1.04502574e-01 -2.37784889e-02  2.73989085e-02  1.55599207e-01
-2.77195079e-03  1.89030655e-02  4.68748361e-02  2.11649723e-02]
```

```

3.29707451e-02 -9.35959592e-02 1.53364509e-01 8.30018222e-02
1.56344399e-01 2.09414542e-01 -1.09019009e-02 -8.27383846e-02
-1.52124390e-01 2.04615861e-01 1.84260547e-01 1.78548589e-01
1.75115541e-02 1.01964595e-02 2.85195210e-03 8.62797499e-02
1.57599911e-01 -6.92393258e-02 2.03184187e-01 3.93796858e-04
2.88663972e-02 2.57268101e-01 -9.62323770e-02 -1.19149141e-01
1.88405007e-01 1.53981045e-01 9.49712843e-02 -7.32565746e-02
-4.71446812e-02 6.39423653e-02 -2.86645442e-02 -7.28903562e-02
2.74329204e-02 1.42536268e-01 3.79656479e-02 -2.90902182e-02
-7.72973225e-02 -3.79364908e-01 1.44835740e-01 2.89077967e-01
1.83294430e-01 -8.90121460e-02 -1.18728027e-01 -3.65421809e-02
1.50683001e-01 5.75584024e-02 6.35507181e-02 -6.93864524e-02
-7.18736127e-02 4.85871844e-02 -7.92968273e-02 -7.60947764e-02
1.75751209e-01 1.56454127e-02 4.42015640e-02 1.00735486e-01
2.36615837e-02 1.41660385e-02 1.73439924e-03 3.85778993e-02
1.47505954e-01 -5.57372011e-02 1.99366614e-01 1.77332927e-02
1.39017720e-02 5.20809088e-03 -3.51771899e-02 -1.91033900e-01
1.33832600e-02 -9.94292200e-02 3.76986302e-02 9.38725099e-03
2.85761002e-02 -4.29738127e-02 7.43370620e-04 -2.47689307e-01
8.02268460e-02 2.27401834e-02 2.83477485e-01 1.24741815e-01
5.13344586e-01 -1.04342841e-01 9.24930423e-02 1.10017419e-01
-3.39711964e-01 -3.63127738e-02 -6.21374734e-02 -9.39397067e-02
-4.27645631e-02 4.19991016e-02 1.27547383e-01 -1.16916738e-01
1.53501838e-01 -1.62372038e-01 -7.45334476e-02 -1.64226457e-01
1.24249734e-01 2.45201990e-01 -2.82593012e-01 -2.52310280e-02
5.01009524e-02 1.28962547e-01 3.16310897e-02 8.67929310e-02
3.72633100e-01 2.05868393e-01 -1.23229228e-01 -2.31928229e-01
1.47817180e-01 -1.47775665e-01 7.49128535e-02 9.20652598e-02
1.92804068e-01 -2.17391551e-01 -6.01991778e-03 8.68222117e-02
4.75134291e-02 -1.32257745e-01 -2.10440829e-02 -1.56973869e-01
-6.19931072e-02 1.25609227e-02 -1.23218983e-01 2.53714114e-01
1.86279014e-01 2.62722641e-01 1.78902242e-02 -1.56033605e-01
9.97112147e-05 8.21634233e-02 -1.82665229e-01 5.08229099e-02
-4.72846329e-02 -1.46754622e-01 5.48403300e-02 1.43918321e-01
1.39083862e-01 2.09148712e-02 -8.22087154e-02 -6.02755025e-02
6.76847994e-02 -1.21093611e-03 -9.16820243e-02 8.49751532e-02
-1.14586517e-01 1.89506263e-01 -1.04682297e-02 -5.79984039e-02
2.53672209e-02 1.11608185e-01 -9.44144577e-02 -3.44396085e-02
6.49087224e-03 1.61137000e-01 -1.84098765e-01 -6.27396954e-03]

```

```

/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:1: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
    """Entry point for launching an IPython kernel.
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:2: DeprecationWarning: Call to deprecated `similarity` (Method will be removed in 4.0.0, use self.wv.similarity() instead).

```

Observation:

-> We have constructed the vector representation of each word

-> Using this model to construct vector representation of each sentence in average word2vec

-> Each word in the model is of 200-dimensions

Nu - Support Vector Classification

AVERAGE WORD2VEC:

TIME BASED SPLITTING OF DATA:

-> Implementing Nu Support Vector Classifier

-> Using both Grid Search and Random Search to determine the best hyper parameters

-> To check performance measure with different hyper parameter values

In [97]:

```
sent_vectors = []
for sent in sentences_list:
    sent_vec = np.zeros(200)
    cnt=0
    for word in sent:
        try:
            vec = w2vmodel.wv[word]
            sent_vec += vec
            cnt += 1
        except:
            pass
    sent_vec /= cnt
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[88888]))
```

100000

200

In [121]:

```
sent_vectors = np.nan_to_num(sent_vectors)
```

In [122]:

```
sample_vect = sent_vectors[0:10000]
sample_score = sorted_w2vec['Score'][0:10000]
```

In [123]:

```
xtrain = sample_vect[0:7000]
xtest = sample_vect[7000:]
ytrain = sample_score[0:7000]
ytest = sample_score[7000:]
```

In [141]:

```
print((xtrain.shape))
print((xtest.shape))
print((ytrain.shape))
```

```
print(ytrain.shape)
print(ytest.shape)
```

```
(7000, 200)
(3000, 200)
(7000,)
(3000,)
```

AVERAGE WORD2VEC: GRID SEARCH TO FIND HYPER PARAMETERS

In [128]:

```
from sklearn.svm import NuSVC
```

In [175]:

```
parameters = {'nu':[0.1,0.2,0.15,0.05]}
```

In [146]:

```
model = GridSearchCV(classifier,param_grid=parameters,scoring='accuracy',cv
=5)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
print(score)
```

```
GridSearchCV(cv=5, error_score='raise',
             estimator=NuSVC(cache_size=200, class_weight=None, coef0=0.0,
                             decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
                             max_iter=-1, nu=0.5, probability=False, random_state=None,
                             shrinking=True, tol=0.001, verbose=False),
             fit_params=None, iid=True, n_jobs=1,
             param_grid={'nu': [0.1, 0.2, 0.15, 0.05]}, pre_dispatch='2*n_jobs',
             refit=True, return_train_score='warn', scoring='accuracy',
             verbose=0)
0.916
```

In [149]:

```
model = NuSVC(nu=0.1)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
print(score)
```

```
NuSVC(cache_size=200, class_weight=None, coef0=0.0,
       decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
       max_iter=-1, nu=0.1, probability=False, random_state=None,
       shrinking=True, tol=0.001, verbose=False)
0.9156666666666666
```

In [150]:

```
model = NuSVC(nu=0.2)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
```

```
print(score)
```

```
NuSVC(cache_size=200, class_weight=None, coef0=0.0,  
      decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',  
      max_iter=-1, nu=0.2, probability=False, random_state=None,  
      shrinking=True, tol=0.001, verbose=False)  
0.916
```

In [151]:

```
model = NuSVC(nu=0.01)  
model.fit(xtrain,ytrain)  
pred = model.predict(xtest)  
score = accuracy_score(ytest,pred)  
print(model)  
print(score)
```

```
NuSVC(cache_size=200, class_weight=None, coef0=0.0,  
      decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',  
      max_iter=-1, nu=0.01, probability=False, random_state=None,  
      shrinking=True, tol=0.001, verbose=False)  
0.8533333333333334
```

AVERAGE WORD2VEC: RANDOM SEARCH TO FIND HYPER PARAMETERS

In [180]:

```
parameters = {'nu': np.linspace(0.0,1,5)}  
parameters
```

Out[180]:

```
{'nu': array([0. , 0.25, 0.5 , 0.75, 1.  ])}
```

In [173]:

```
classifier = NuSVC()
```

In [178]:

```
model = RandomizedSearchCV(classifier,param_distributions=parameters,cv=5,n  
_iter=4)  
model.fit(xtrain,ytrain)
```

Out[178]:

```
RandomizedSearchCV(cv=5, error_score='raise',  
                  estimator=NuSVC(cache_size=200, class_weight=None, coef0=0.0,  
                                decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',  
                                max_iter=-1, nu=0.5, probability=False, random_state=None,  
                                shrinking=True, tol=0.001, verbose=False),  
                  fit_params=None, iid=True, n_iter=4, n_jobs=1,  
                  param_distributions={'nu': [0.1, 0.2, 0.15, 0.05]},  
                  pre_dispatch='2*n_jobs', random_state=None, refit=True,  
                  return_train_score='warn', scoring=None, verbose=0)
```

In [181]:

```
pred = model.predict(xtest)  
score = accuracy_score(ytest,pred)  
print(score)
```

0.916

In [184]:

```
model = NuSVC(nu=0.01)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
print(score)
```

```
NuSVC(cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
      max_iter=-1, nu=0.01, probability=False, random_state=None,
      shrinking=True, tol=0.001, verbose=False)
0.8533333333333334
```

In [185]:

```
model = NuSVC(nu=0.1)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(model)
print(score)
```

```
NuSVC(cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
      max_iter=-1, nu=0.1, probability=False, random_state=None,
      shrinking=True, tol=0.001, verbose=False)
0.9156666666666666
```

Observation:

AVERAGE word2VEC:

-> By using Grid Search with NuSVC the best values of hyper parameters were nu = 0.5

-> By using Random search with NuSVC the best values of hyper parameters were nu = 0.5

-> The accuracy by grid search is 91.6

-> The accuracy of random search is 91.6

CONCLUSION:

RBF SVM:

BAG OF WORDS:

-> Implemented RBF SVM with Grid Search and random Search

-> By using Grid Search with "RBF" as kernel the best values of hyper parameters were C = 1 and gamma = 3

of hyper parameters were $C = 1$ and $\gamma = 0$

-> By using Random search with "RBF" as kernel the best values of hyper parameters were $C = 3$ and $\gamma = 2$

-> The accuracy by grid search is 84.19

-> The accuracy of random search is 84.24

-> Measured accuracy with different combinations of hyper parameters

LINEAR SVM:

TFIDF:

-> Implemented Linear SVM with Grid Search and random Search

-> By using Grid Search with Linear SVM the best values of hyper parameters were $C = 0.001$

-> By using Random search with Linear SVM the best values of hyper parameters were $C = 8$

-> The accuracy by grid search is 89.36

-> The accuracy of random search is 87.14

-> Measured accuracy with different combinations of hyper parameters

NuSVM:

AVERAGE WORD2VEC:

-> Implemented Nu SVM with Grid Search and random Search

-> By using Grid Search with NuSVC the best values of hyper parameters were $\nu = 0.5$

-> By using Random search with NuSVC the best values of hyper parameters were $\nu = 0.5$

-> The accuracy by grid search is 91.6

-> The accuracy of random search is 91.6

-> Measured accuracy with different combinations of hyper parameters