## 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- unqiue identifier for the user

                        ProfileName

                        HelpfullnessNumerator- number of users who fou
nd the review helpful

                        HelpfulnessDenominator- number of users who in
dicated 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 review we cla
ssify them into positive and negative

                  Number of reviews: 568,454
```

## objective:

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

-> Converting the text data to vectors by using word2vec,Average wor
d2vec

-> Applying Decision trees do determine the best depth of tree havin
g accuracy as the check measure
```

```
-> Here we can use both random based splitting of data or time based
splitting of data

-> Since it is the data of reviews they may change overtime due to m
odifications of products

-> The accurancy which we obtained by random based splitting may cha
nge for the future data

-> We can assure about the accurancy obtained in time base splitting
for unseen future data also.

->Random based splitting is possible for every dataset, but for time
based splitting there should time attribute
```

Importing the required libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as mp
import seaborn as s
import sqlite3
import nltk
import string
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

->loading the data and information about the data

-> Shape of the data

-> Dimensionality of the data

-> Attributes if the data

In [2]:

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

```
(525814, 10)
2
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

Removing the Duplicates from the data

In [3]:

In [3]:

```
####function to categorise rating into positive and negatives
def change(n):
    if n>3:
        return 'positive'
    return 'negative'

rating = data['Score']
####take the ratings
rating = rating.map(change)
#####apply function change on ratings column
data['Score'] = rating
####updating the column with positive and negatives
data.head(6)
##### head with first 6 elements in data
```

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 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | 1 |
| **3** | 4 | B000UA0QIQ | A395BORC6FGVXV | Karl | 3 | 3 |
| **4** | 5 | B006K2ZZ7K | A1UQRSCLF8GW1T | Michael D. Bigham "M. Wassir" | 0 | 0 |

| 5 | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfulness |
|---|---|---|---|---|---|---|
| | 6 | B006K2ZZ7K | ADT0SRK1MGOEU | Twoapennything | 0 | 0 |

Data Cleaning:Removing Duplicates

In [4]:

```
user = pd.read_sql_query("""SELECT * FROM Reviews WHERE UserId= "AR5J8UI46C
URR" ORDER BY ProductId """,con)
print(user)
```

```
        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  
\
0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2   

1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2   

2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2   

3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2   

4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2   


   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600   
1                       2      5  1199577600   
2                       2      5  1199577600   
3                       2      5  1199577600   
4                       2      5  1199577600   

                            Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS   
1  LOACKER QUADRATINI VANILLA WAFERS   
2  LOACKER QUADRATINI VANILLA WAFERS   
3  LOACKER QUADRATINI VANILLA WAFERS   
4  LOACKER QUADRATINI VANILLA WAFERS   

                                                Text  
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...  
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...  
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...  
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...  
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...  
```

Observation:

```
-> Here we can see that for the same time span we got five reviews,
practically which is not possible

->This happened because when the user given review for a product it
is applied to all the flavors of the product
```

In [5]:

```
sorteddata = data.sort_values('ProductId',axis=0,ascending=True,inplace=Fal
se,kind='quicksort',na_position='last')
finaldata = sorteddata.drop_duplicates(subset={"UserId","ProfileName","Time
","Text"},keep='first',inplace=False)
```

Information about the modified data:

```
-> Shape of the data

-> Dimensionality of the data

-> Attributes if the data

-> Sample of modified data
```

In [6]:

```
print(finaldata.shape)
print(finaldata.ndim)
print(finaldata.columns)
print(finaldata.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  A2IW4PEEKO2R0U                         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...
```

CONSTRUCTING VECTOR REPRESENTATION OF EACH IN THE DATA BY USING WORD2VEC

In [7]:

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

-> Importing the required libraries

-> Functions to clean the sentences

-> Constructing the word2vec from the sample subset data

In [8]:

```
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 [9]:

```
sorted_w2vec = finaldata.sort_values("Time",axis=0,ascending=True,kind='qui
cksort',na_position='last',inplace=False)
```

Information about the sorted data:

```
-> Shape of the data

-> Dimensionality of the data

-> Attributes if the data

-> Sample of modified data
```

In [10]:

```
print(sorted_w2vec.shape)
print(sorted_w2vec.ndim)
print(sorted_w2vec.columns)
print(sorted_w2vec.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
138683  150501  0006641040   AJ46FKXOVC7NR     Nicholas A Mesiano
417839  451856  B00004CXX9   AIUWLEQ1ADEG5       Elizabeth Medina
```

```
346055   374359   B00004CI84   A344SMIA5JECGM            Vincent P. Ross
417838   451855   B00004CXX9    AJH6LUC1UT1ON  The Phantom of the Opera

        HelpfulnessNumerator  HelpfulnessDenominator     Score        Time  \
138706                     0                       0  positive   939340800
138683                     2                       2  positive   940809600
417839                     0                       0  positive   944092800
346055                     1                       2  positive   944438400
417838                     0                       0  positive   946857600

                                                  Summary  \
138706                            EVERY book is educational
138683  This whole series is great way to spend time w...
417839                                 Entertainingl Funny!
346055                              A modern day fairy tale
417838                                           FANTASTIC!

                                                     Text
138706  this witty little book makes my son laugh at l...
138683  I can remember seeing the show when it aired o...
417839  Beetlejuice is a well written movie ..... ever...
346055  A twist of rumplestiskin captured on film, sta...
417838  Beetlejuice is an excellent and funny movie. K...
```

In [11]:

```
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 [12]:

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

```
364173
<class 'list'>
```

In [13]:

```
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 [14]:

```
print(w2vmodel.most_similar("where"))
```

```
print(w2vmodel.similarity("where",'when'))
print(w2vmodel.wv['what'])
```

```
[('wherever', 0.4594731330871582), ('what', 0.45790600776672363), ('when',
0.4426317811012268), ('atlanta', 0.417472779750824), ('everywhere', 0.40873
72124195099), ('somewhere', 0.4012754559516907), ('why',
0.3978917002677917 5), ('miami', 0.3965842425823211 7), ('nyc',
0.3958161771297455), ('florida', 0.39461496472358704)]
0.44263175693579765
[ 1.6331531  -0.79845667 -0.45216376  0.42539343 -0.33658674  0.59380037
 -0.5263159   0.5842386   1.466138   -0.3119752  -0.21973877  0.8555864
 -3.6795366   0.8617518   3.0770397  -0.11212743 -0.72588176  2.4425774
 -1.5533571  -0.59036565  1.7174312  -0.77355695  1.3786777   0.6814901
  0.6741636   0.8354629  -1.707923    1.3012035   0.9157588   3.9978266
 -0.631389    0.1697004  -2.1428678  -1.0298405   0.60593456  1.6675051
  0.5104288   1.8326571   1.6684786   1.7801542  -1.6682345   0.14508161
  1.2019076   1.3818147   1.8297708   0.06275181 -0.2328718   0.8112764
 -4.7815714   2.254828   -1.2480936  -0.20308697 -1.0630426   0.92793036
  1.8030949  -1.7717417  -1.681844    0.18603978  0.7955026   0.89024246
  1.9989334   0.9757756   1.3316698   4.8155875  -1.3277873   3.258908
  1.1285194  -0.2615028   1.6847275   0.1742869   1.8900928  -0.19013865
  2.279615   -2.3754504  -0.38275328 -1.6026828  -2.2104282  -2.6147068
 -1.3613904  -0.75397664 -1.3671336  -2.3176205   1.3250021  -0.8424476
 -2.2235625   1.9410805  -2.5142384  -1.0206442   0.41692322  0.11198874
  0.03515576  1.5786471   0.19432983 -2.0514994  -1.4727927   0.40654066
  0.88286346 -2.428184    1.0518938  -1.2216723  -1.0388702  -0.623032
 -0.41029868 -1.2505282   0.70760673  1.0133395  -0.3811332  -1.3858659
 -0.36918342 -2.7233331   0.9510346  -0.35236558  0.97679293  0.7148403
 -0.78331006 -0.61408746 -0.3376292  -0.27254063  0.555511   -1.062949
  1.2672429  -0.8551705   0.07587209 -2.0158272  -0.322324    0.6007844
 -0.9152724   1.457027    2.5418932   2.4028869   1.0896951  -0.24308997
 -2.1541197  -1.385261    1.7806646  -1.4152241   1.0486333  -1.5356334
 -1.763022    1.7497728   1.2591652   1.1417272  -1.5481972  -2.2717617
  1.3790187  -1.8005618  -0.06050143 -2.5310838  -1.4711559   2.5667489
  1.0004866  -1.3947448  -1.7634138  -0.58484435  0.7129905   2.835419
  0.45729092  0.4322272   0.30003986 -2.8693535   1.5222995  -0.45841023
 -1.6101103  -1.1513519   1.583578   -1.9039611  -2.0539722   0.796204
  0.5662639  -3.0849988   1.1563088   0.59110135 -1.1242639   1.0625143
  1.8337064   0.07630508 -0.00891196 -0.1754792  -0.7249214   2.6913538
 -3.177442   -0.28687838 -3.4646447  -0.68664604 -2.9991336  -1.0978917
  1.0261513  -1.767935   -1.0438273   0.12836058 -0.5314205  -1.8073746
 -1.30182     1.3502879  -2.0460868  -1.7149078  -3.5365427   0.15722775
  0.12164022 -3.0982144 ]
```

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

Observation:

```
-> We have constructed the vector representation of each word

-> Using this model to construct vector representation of each sente
```

nce in average word2vec and tfidf-word2vec

## AVERAGE WORD2VEC

```
-> Here i am using the word2vec model to construct vector
representation of each sentence
```

In [15]:

```
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[364000]))
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/ipykernel_launcher.py:13: RuntimeWarning: invalid value
encountered in true_divide
  del sys.path[0]
```

```
364173
200
```

In [16]:

```
np.isnan(sent_vectors).any()
```

Out[16]:

```
True
```

In [17]:

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

In [18]:

```
np.isnan(sent_vectors).any()
```

Out[18]:

```
False
```

In [19]:

```
sent_vectors.shape
```

Out[19]:

```
(364173, 200)
```

In [20]:

```
xtrain = sent_vectors[0:250000]
xtest = sent_vectors[250000:]
ytrain = sorted_w2vec['Score'][0:250000]
ytest = sorted_w2vec['Score'][250000:]
```

In [21]:

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

```
(250000, 200)
(114173, 200)
(250000,)
(114173,)
```

In [43]:

```
l = np.arange(1,10,1)
l
```

Out[43]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [25]:

```
from sklearn.cross_validation import cross_val_score
```

In [28]:

```
cross_validation_scores = []
for d in l:
    model = DecisionTreeClassifier(criterion='gini',max_depth=d,
min_samples_split=10)
    score = cross_val_score(model,xtrain,ytrain,cv=3,scoring='accuracy')
    cross_validation_scores.append(score.mean())
```

In [30]:

```
error = [1 - x for x in cross_validation_scores]
print(error)
print(cross_validation_scores)
```

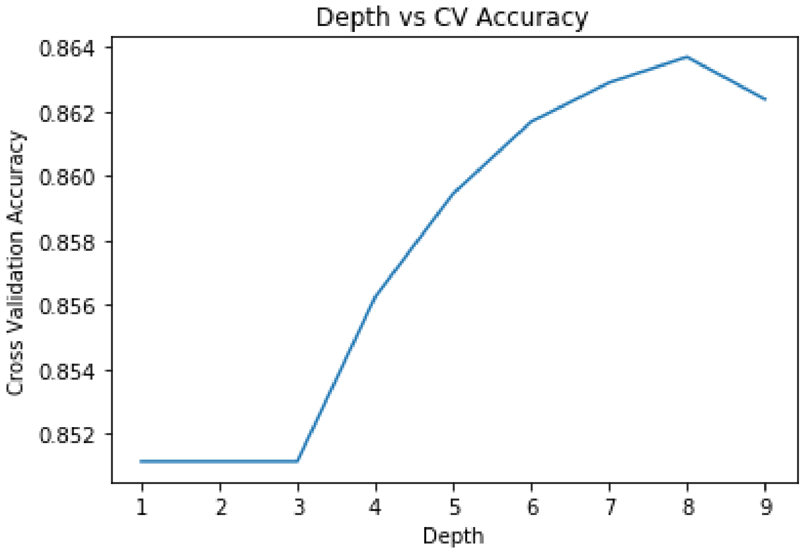
```
[0.14885599995505316, 0.14885599995505316, 0.14885599995505316,
0.14375999109033, 0.14055999768197314, 0.13833199467366386,
0.13711999659352403, 0.13632399374539939, 0.13762799550558047]
[0.8511440000449468, 0.8511440000449468, 0.8511440000449468,
0.85624000890967, 0.8594400023180269, 0.8616680053263361,
0.862880003406476, 0.8636760062546006, 0.8623720044944195]
```

In [31]:

```
mp.plot(l,cross_validation_scores)
mp.xlabel('Depth')
mp.ylabel('Cross Validation Accuracy')
mp.title("Depth vs CV Accuracy")
```

Out[31]:

```
Text(0.5,1,'Depth vs CV Accuracy')
```

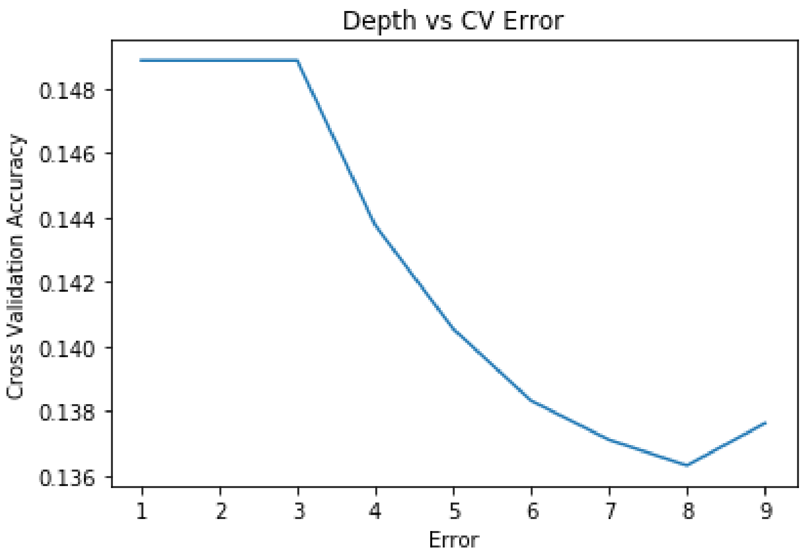



In [33]:

```
mp.plot(l,error)
mp.xlabel('Error')
mp.ylabel('Cross Validation Accuracy')
mp.title("Depth vs CV Error")
```

Out[33]:

```
Text(0.5,1,'Depth vs CV Error')
```




In [35]:

```
best_d = l[error.index(min(error))]
print("the best value of d is {}".format(best_d))
```

```
the best value of d is 8
```

In [36]:

```
model = DecisionTreeClassifier(criterion='gini',max_depth=best_d, min_sampl
es_split=10)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
```

```
score = accuracy_score(ytest,pred)
print(score)
```

```
0.8490185945889133
```

Observation:

    -> The optimal depth of the decision tree is 8

    -> The accuracy with depth of 8 is 84.90

TFIDF WORD2VEC:

    -> Here i am using the word2vec model to construct vector
    representation of each sentence

In [22]:

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

In [23]:

```
data.shape
```

Out[23]:

```
(364173, 10)
```

In [24]:

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

In [25]:

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

In [26]:

```
tfid_vect = tfid.fit_transform(data['Text'].values)
```

In [27]:

```
tfid_vect.shape
```

Out[27]:

```
(364173, 2910206)
```

In [28]:

```
tfidf_feat = tfid.get_feature_names()
print(len(tfidf_feat))
```

```
2910206
```

In [30]:

```
tfidf_feat = tfid.get_feature_names()
tfidf_sent_vectors = [];
row=0;
for sent in sentences_list:
    sent_vec = np.zeros(200)
    sum =0;
    for word in sent:
        try:
            vec = w2v_model.wv[word]
            tfidf = tfid_vect[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            sum += tf_idf
        except:
            pass
    sent_vec /= sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: RuntimeWarning: invalid value encountered in true_divide
  from ipykernel import kernelapp as app
```

In [31]:

```
print(len(tfidf_sent_vectors))
print(len(tfidf_sent_vectors[300000]))
```

```
364173
200
```

In [32]:

```
type(tfidf_sent_vectors)
```

Out[32]:

```
list
```

In [35]:

```
tfidf_sent_vectors = np.nan_to_num(tfidf_sent_vectors)
```

In [37]:

```
np.isnan(tfidf_sent_vectors).any()
```

Out[37]:

```
False
```

In [39]:

```
xtrain = tfidf_sent_vectors[0:250000]
xtest = tfidf_sent_vectors[250000:]
ytrain = data['Score'][0:250000]
ytest = data['Score'][250000:]
```

In [40]:

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

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

```
(250000, 200)
(114173, 200)
(250000,)
(114173,)
```

In [41]:

```
from sklearn.cross_validation import cross_val_score
```

```
/Users/vthumati/anaconda3/lib/python3.6/site-
packages/sklearn/cross_validation.py:41: DeprecationWarning: This module wa
s deprecated in version 0.18 in favor of the model_selection module into wh
ich all the refactored classes and functions are moved. Also note that the
interface of the new CV iterators are different from that of this module. T
his module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [45]:

```
cross_validation_scores = []
for d in l:
    model = DecisionTreeClassifier(criterion='gini',max_depth=d,
min_samples_split=10)
    score = cross_val_score(model,xtrain,ytrain,cv=3,scoring='accuracy')
    cross_validation_scores.append(score.mean())
```

In [46]:

```
error = [1 - x for x in cross_validation_scores]
print(error)
print(cross_validation_scores)
```

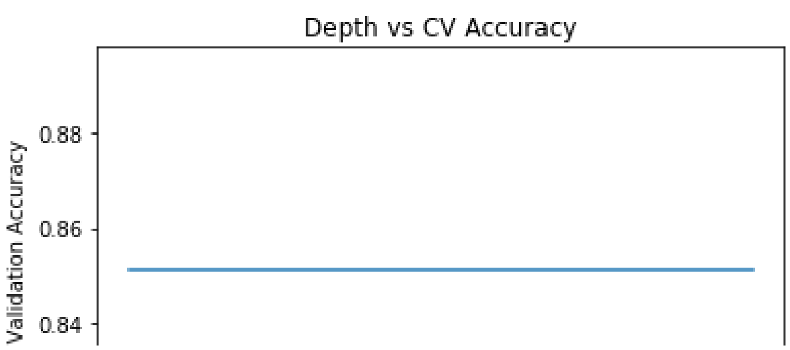
```
[0.14885599995505316, 0.14885599995505316, 0.14885599995505316,
0.14885599995505316, 0.14885599995505316, 0.14885599995505316,
0.14885599995505316, 0.14885599995505316, 0.14885599995505316]
[0.8511440000449468, 0.8511440000449468, 0.8511440000449468,
0.8511440000449468, 0.8511440000449468, 0.8511440000449468,
0.8511440000449468, 0.8511440000449468, 0.8511440000449468]
```

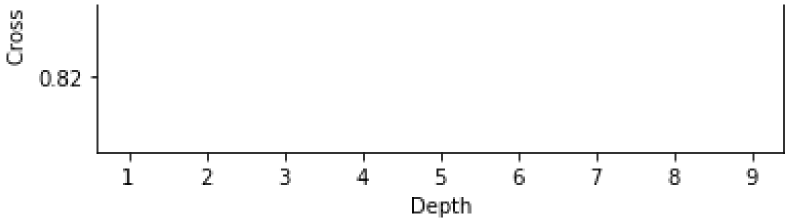
In [47]:

```
mp.plot(l,cross_validation_scores)
mp.xlabel('Depth')
mp.ylabel('Cross Validation Accuracy')
mp.title("Depth vs CV Accuracy")
```

Out[47]:

```
Text(0.5,1,'Depth vs CV Accuracy')
```



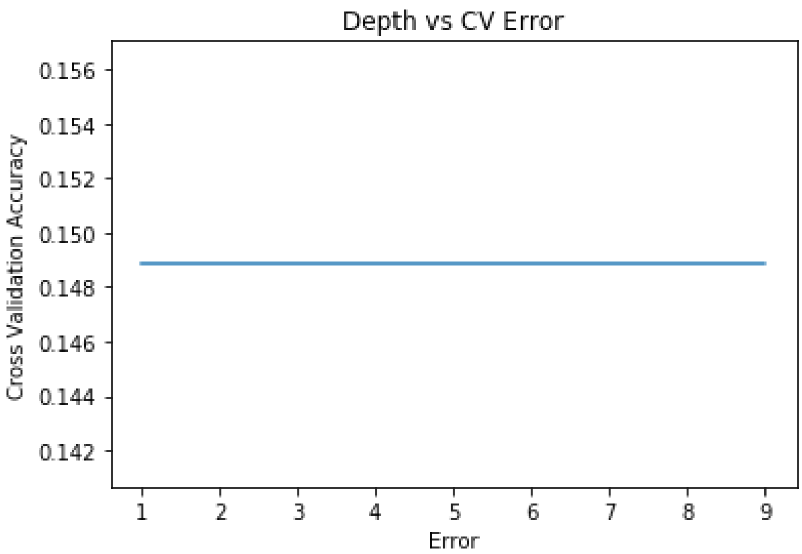



In [48]:

```
mp.plot(l,error)
mp.xlabel('Error')
mp.ylabel('Cross Validation Accuracy')
mp.title("Depth vs CV Error")
```

Out[48]:

```
Text(0.5,1,'Depth vs CV Error')
```




In [49]:

```
best_d = l[error.index(min(error))]
print("the best value of d is {}".format(best_d))
```

```
the best value of d is 1
```

In [51]:

```
model = DecisionTreeClassifier(criterion='gini',max_depth=best_d, min_sampl
es_split=10)
model.fit(xtrain,ytrain)
pred = model.predict(xtest)
score = accuracy_score(ytest,pred)
print(score)
```

```
0.8257381342348892
```

Observation:

-> The optimal depth of the decision tree is 1

-> The accuracy with depth of 1 is 82.57

CONCLUSION:

```
-> Determined the optimal depth with the help of cross validation

AVERAGE WORD2VEC:

    -> The optimal depth of the decision tree is 8

    -> The accuracy with depth of 8 is 84.90

TFIDF WORD2VEC:

    -> The optimal depth of the decision tree is 1

    -> The accuracy with depth of 1 is 82.57
```