OverAll Procedure-

Step 1: Crawling over the Internet, and saving information about retail companies into training.txt file.

Step 2: Cleaning the data-The data is in form of paragraphs and sentences. It is tokenized and from that only those tokens are chosen which potentially contains information about the vendor. For example, only the Nouns are kept and words which are tagged under Name, Organization, Location or Entity are kept.

Step 3: At this step, the cleaned words are changed into vectors via Word2Vec. Word2Vec changes each word into a vector containing an array of numbers such that mathematical computations like clustering can be done on textual data. Now these words are used for forming Clusters using K Means Clustering.

Step 4: Now we input a vendor (whose details we need to find out), We crawl over the training.txt file and extract only those paragraphs which are pertaining to the input vendor. We do it via NER Tagger.We check the Organization for each paragraph in our training set and in which paragraphs the (Organization==Input Vendor Name) and if this has happened more than thrice it is considered as a suitable paragraph for the Vendor.This is our Test Data

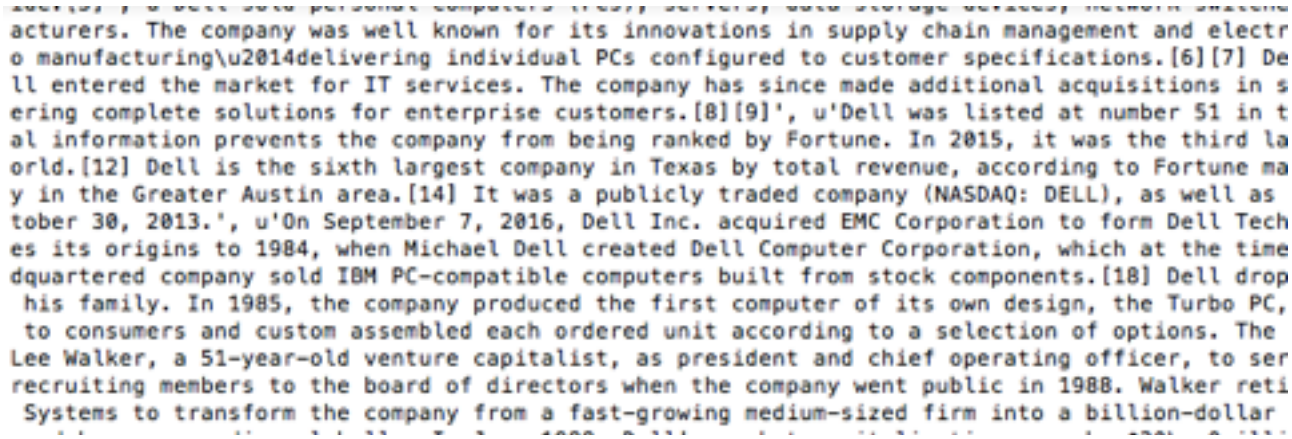
Step 5: Now we use our Kmeans model created in Step 3 and determine to which cluster the words in our Test Data belongs to. Thus we create a Cluster frequency matrix like Cluster 1:5 words, Cluster 2:1 words,Cluster 3:0 words. Now we select the top 7 words (which has occurred maximum number of times in the Test Data) from the cluster containing maximum number of words and top 3 words from the 2nd max cluster.

Code Explained in Details with Screenshot of Outputs displayed, ran on small sample of Data and Output predicted for Vendor Dell:-

1. **crawling.py** (Used for Web Crawling to create the Train Data)

```
from bs4 import BeautifulSoup   #BeautifulSoup is a library for data crawling
import urllib

#Paste the URL here which you want to crawl the data from
thisurl = "https://en.wikipedia.org/wiki/Hewlett-Packard"
soup = BeautifulSoup(urllib.urlopen(thisurl).read(),'html.parser')

#Find the relevant data in the document
data=''.join([x.text for x in soup.find(id='mw-content-text').find_all('p')]).encode('utf-8');

#File is opened in Append mode so that it is appended everytime a URl is crawled
with open("trainingdata.txt", "a") as myfile:
myfile.write(data)
myfile.close()
```

Output: (training.txt)

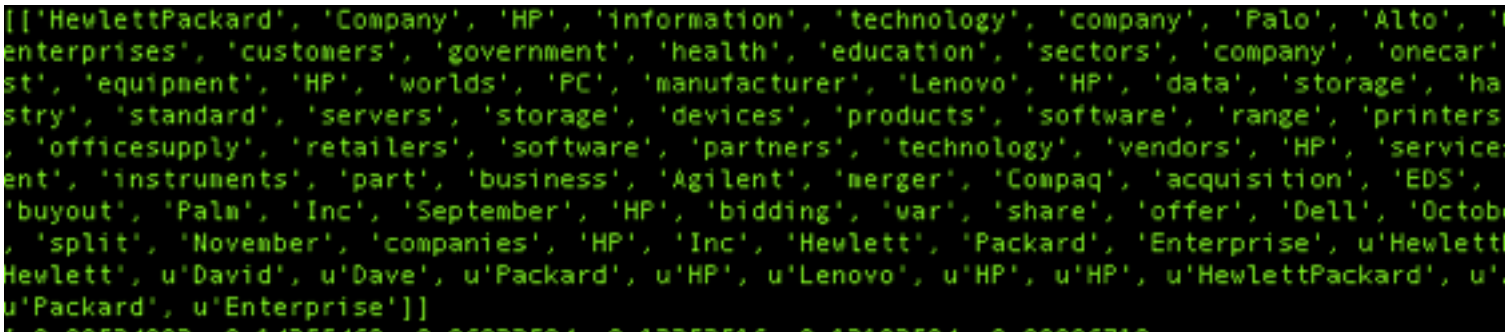


2. **cleaning.py** (Used for cleaning the paragraphs and changing them into useful tokens)

```
import pandas as pd
from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.tag import pos_tag
from nltk.tag import StanfordNERTagger
import nltk.data
import re

#For reading sentences from the training module and saving it to a list Query1
query1=[]
with open("Desktop/trainingdata.txt", "r") as myfile:
query1.append(myfile.read())
myfile.close()

#Tokenizing sentences into words and removing punctuations
def remove_punctuations(text):
wordlist=[]
sentences=sent_tokenize(text)
for sentence in sentences:
        words=[word for word in sentence.split()]
        words=[re.sub(r'[^\w\s]','',word) for word in words]
        wordlist.append(words)
return wordlist


#Keeping only Nouns from the sentence
def descriptive_words(words):
```

```
tagged_word=[]
meaningful_words=[]
tags=['NN','NNS','NNP']

for word in words:
        tagged_word.append(pos_tag(word))
        for words in tagged_word:
                for word in words:
                        if word[1] in tags:
                                meaningful_words.append(word[0])

return meaningful_words


#Using NER Tagger to find Name, Organisation, Entity
def remove_names(text):
        meaningful_words=[]
        tagged_word=[]
        tags=['LOCATION', 'ORGANIZATION', 'PERSON']
        st = StanfordNERTagger('english.all.3class.distsim.crf.ser.gz')

        for sentence in text:
                tagged_word.append(st.tag(sentence))

        for words in tagged_word:
                for word in words:
                if word[1] in tags:
                        meaningful_words.append(word[0])
        return meaningful_words




#Using the above functions for Cleaning of Sentences in Query1
words_req=[]

for query in query1:
#Removing Punctuations and changing to lower space
letters_only = remove_punctuations(query)

#5 Read only NOuns,Pronouns,interjections (descriptive words)
meaningful_words=descriptive_words(letters_only)


#6 Keeping Time, Location, Organization, Person, Money, Percent, Date using NER
removed_words=remove_names(letters_only)


#6.We would not Stem our data because we would convert our data into vectors in the next
        step, Stemming them would loose the information saved in them
stemmed_words=[words for words in meaningful_words+removed_words]
words_req.append(stemmed_words)
```

Output - After Cleaning every paragraph changes into the form of

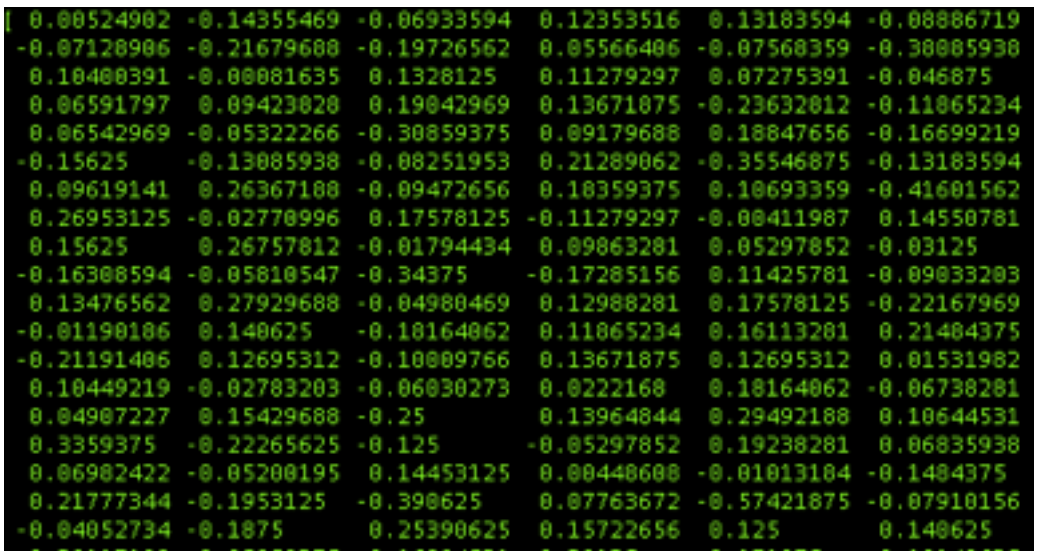


4. **clustering.py** (Changing words into vectors and then applying K Means Clustering on them)

```
from gensim import models
from sklearn.cluster import Means

#Loading Googles PreTrained Vector Model
wordvec = models.Word2Vec.load_word2vec_format('GoogleNews-vectors-        nega
tive300.bin', binary=True)
vecs=[]

#Changing the words into Vectors so that Clusters can be formed out of them
for sentence in words_req:
        for words in sentence:
                word=wordvec[words]
                vecs.append(word)

#Applying Clustering to it
kmeans = KMeans(init='k-means++', n_clusters=10, n_iter=100)
kmeans.fit(vecs)
```

Output-

Word converted to a vector (this is a vector for a single word)



Clusters formed on these vectors

```
Clustering Done
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_init=10,
    n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
    verbose=0)
```

5. **predict.py** (First find out the paragraphs which points out to your input vendor as your train data, then identify to which clusters your train data belongs to.)

```
from gensim import models
from sklearn.cluster import KMeans
import cleaning.py

#Searching for only those Sentences from the training module which belongs to the
 vendor we are searching for DELL in this case and save it in Query3
query2=[]
with open("Desktop/trainingdata.txt", "r") as myfile:
query2.append(myfile.read())
myfile.close()

query3=[]
for query in query2:
letters_only = remove_punctuations(query)
removed_words=remove_names(letters_only)
if "DELL" in removed_words:
        query3.append(query)

#Now Predicting to which Cluster the words describing the Vendor belongs to
cluster_data={'0':0,'1':0,'2':0,'3':0,'4':0,'5':0,'6':0,'7':0,'8':0,'9':0}
cluster_words={'0':[],'1':[],'2':[],'3':[],'4':[],'5':[],'6':[],'7':[],'8':[],'9':[]}

for pos,vec in enumerate(vecs):
        cluster_no=kmeans.predict([vec])
        cluster_data[str(cluster_no[0])]+=1
        cluster_words[str(cluster_no[0])].append(words[pos])

#find cluster which contains maximum number of words, print the top 7 (which oc
cured the most in the training set for that vendor) from that cluster

max_cluster=max(cluster_data, key=cluster_data.get)
imp_words_1=set(cluster_words[max_cluster])

#Findind the 2nd max cluster,print the top 3 (which occured the most in the training
set for that vendor) from that cluster

cluster_data[max_cluster]=0
max_cluster=max(cluster_data, key=cluster_data.get)
imp_words_2=set(cluster_words[max_cluster])
```

Output-
Cluster Matrix formed

```
Cluster matrix formed
{'1': 5, '0': 0, '3': 0, '2': 5, '5': 0, '4': 0, '7': 0, '6': 0, '9': 0, '8': 0}
{'1': [u'deal', u'Hardvar', u'Softvar', u'Hardvar', u'Softvar'], '0': [], '3': [], '2': [u'Dell', u'Laptop', u'
```

Final Predicted Output for Vendor name== “DELL”

```
set([u'Hardwar', u'Softwar', u'deal'])
set([u'Laptop', u'Dell'])
```