In [1]:

```
%config IPCompleter.greedy=True
```

In [2]:

```
import os
from string import punctuation as puncs
import string
import math
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:

```
## define path of your dataset folder here:
path = 'C:\\Corpus\\small sample\\'
```

In [4]:

```
def GetTextFilesPaths():
    txtFiles = []

    for fileName in os.listdir(path):
        if fileName.endswith('.txt'):
            txtFiles.append(path+fileName)
    return txtFiles
```

In [5]:

```
from nltk.corpus import stopwords
def removeStopWords(cleanwordlistofdoc):
    stop_words = set(stopwords.words('english'))

    listStopWordsRemoved = []
    for w in cleanwordlistofdoc:
        if w not in stop_words:
            listStopWordsRemoved.append(w)
    return listStopWordsRemoved
```

In [6]:

```
def DocDataCleaningWordList_func(docPath):
    """
    This function takes string doc path and returns list of word
    """
    file = open(docPath,mode='r')
    completeText = file.read()
    file.close()
    completeText=completeText.lower()
    words_List=wordList(completeText)
    words_List=removePuncs(words_List)
    words_List=removeStopWords(words_List)
    return words_List
```

In [7]:

```
## function to read text and return list of words
def wordList(doc):
    """
    This function should take text which is a string object and return all
    the list of words in it in the same sequece as they appear in the document
    NOTE: you have to make sure your text has same case (lower/upper)
    """
    sList=[]
    doc=doc.lower()
    sList= doc.split(" ")
    return sList
```

In [8]:

```
###  function to remove puntuation marks from words
# import string.maketrans as textfilter
from string import punctuation as puncs
def removePuncs(wordList):
    """
    This function will take a list of words, iterate over the list and remove pun
    """
#     print('punctuation marks are: ', puncs)
    newWordList=[]
    mappingDict=str.maketrans({key: None for key in puncs})
    escapeDict= str.maketrans({'\n':None})
    NewDict= dict((list(mappingDict.items()) + list(escapeDict.items())))
#     print(NewDict)
#     print(escapeDict)
    for w in wordList:
        word=w.translate(NewDict)
        newWordList.append(word)
    return newWordList
```

In [9]:

```
### function to calculate term frequency in the doc
def termFrequencyInDoc(wordList):
    """
    This function should take a list of words as input argument, and output a dic
    each word that appears in the document is key in the dictionary and it's valu
    """
    termFrequency_dic={}
    for w in wordList:
        if w in termFrequency_dic.keys():
            termFrequency_dic[w]+=1
        else:
            termFrequency_dic[w]=1
    return termFrequency_dic
```

In [10]:

```
## function to calculate word Document frequency
def DocFreqOfWord_func(DocDict):
    """
    DocFrequency:Num of docs in which word is found
    Argument:Dictionary that has filename as key and termFrequencyDict as value
    Returns:DocFreqDict that has word as key and its doc_count as value
    """
    DocFrequencyDic={}

    for k, v in DocDict.items():
        for key in v.keys():
            if(key in DocFrequencyDic.keys()):
                DocFrequencyDic[key]+=1
            else:
                DocFrequencyDic[key]=1

    return DocFrequencyDic
```

In [11]:

```
## you should be well versed in syntax of creating functions by now !!
## construct a function named inverseDocFre() that takes dictionary returned from
## and outputs inverse document frequency of each word. You can do it!
import math
def InverseDocFreq_func(DocFreqOfWordDict,totalNumofDoc):
    """
    Argument:Dictionary that has word as key and its value is docFrequency
    Returns:InverseDocFreqDict that has word as key and its IDF as value
    """
    totalNumofDoc+=1

    inverseDocFreqDict={}

    for k,v in DocFreqOfWordDict.items():
        Idf= totalNumofDoc/v
        Idf=math.log10(Idf)
        inverseDocFreqDict[k]=Idf


    return inverseDocFreqDict
```

In [12]:

```
import math
def InverseDocFreqlogbase_func(DocFreqOfWordDict,totalNumofDoc,logbase):
    """
    Argument:Dictionary that has word as key and its value is docFrequency
    Returns:InverseDocFreqDict that has word as key and its IDF as value
    """
    totalNumofDoc+=1

    inverseDocFreqDict={}

    for k,v in DocFreqOfWordDict.items():
        Idf= totalNumofDoc/v
        Idf= math.log(Idf,logbase)
        inverseDocFreqDict[k]=Idf


    return inverseDocFreqDict
```

In [13]:

```
def calculateTFIDF(termFreqDict,IdfDict):
    tfidfDict={}
    for k,v in termFreqDict.items():
        wordIdfValue=IdfDict[k]
        wordTermFreq=v
        tfidf=wordIdfValue*wordTermFreq
#         print(";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;")
#         print(k)
#         print(wordIdfValue)
#         print(wordTermFreq)
#         print(tfidf)
#         print(";;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;")
        tfidfDict[k]=tfidf

    return tfidfDict
```

In [14]:

```
docDictionary={}
docFreqofWordDict={}
IDFdict={}
docTfIdfDict={}
TextFilesPaths=[]
```

In [15]:

```
### function to calculate tf-idf for everyword in doc
## this is your main function which calls the function above in appropriate fasio
def tfidf():
    """
    This function takes list of documents it calls the function wordList to split
    stopwords and punctuation marks from them, then calls termFrequencyInDoc() us
    dictionary of vocabulary using the function wordDocFre(), it then should call
    it then outputs a list of dictionary, where each document corresponds to one 
    values should be tf-idf score
    """


    global docDictionary
    global docFreqofWordDict
    global IDFdict
    global docTfIdfDict
    global TextFilesPaths

    TextFilesPaths=GetTextFilesPaths()

    for filePath in TextFilesPaths:
        words_List=DocDataCleaningWordList_func(filePath)
        termFrequencyDict=termFrequencyInDoc(words_List)
        docDictionary[filePath] = termFrequencyDict

    totalNumOfDocs=len(TextFilesPaths)

    docFreqofWordDict=DocFreqOfWord_func(docDictionary)
    IDFdict=InverseDocFreq_func(docFreqofWordDict,totalNumOfDocs)




    for fileNameKey,TermFreqDict in docDictionary.items():
        docTfIdfDict[fileNameKey]=calculateTFIDF(TermFreqDict,IDFdict)
#         print(fileNameKey)
#         print(docTfIdfDict[fileNameKey])
#         print("--------------------")

    return docTfIdfDict
```

In [16]:

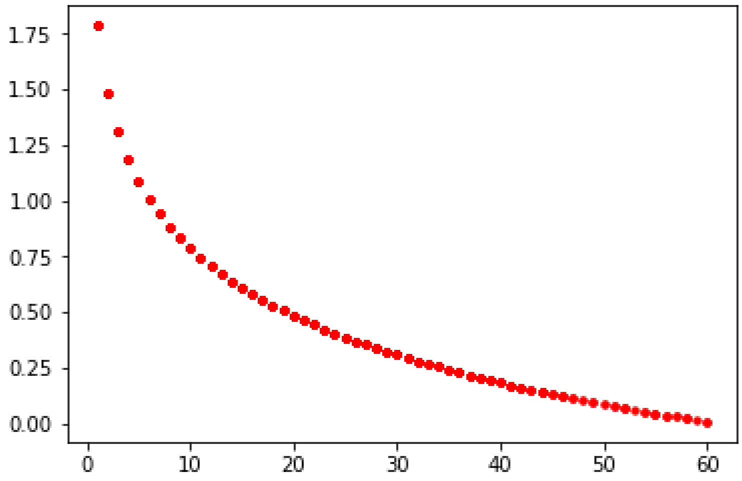
```
tfidf()
print('')
```

In [17]:

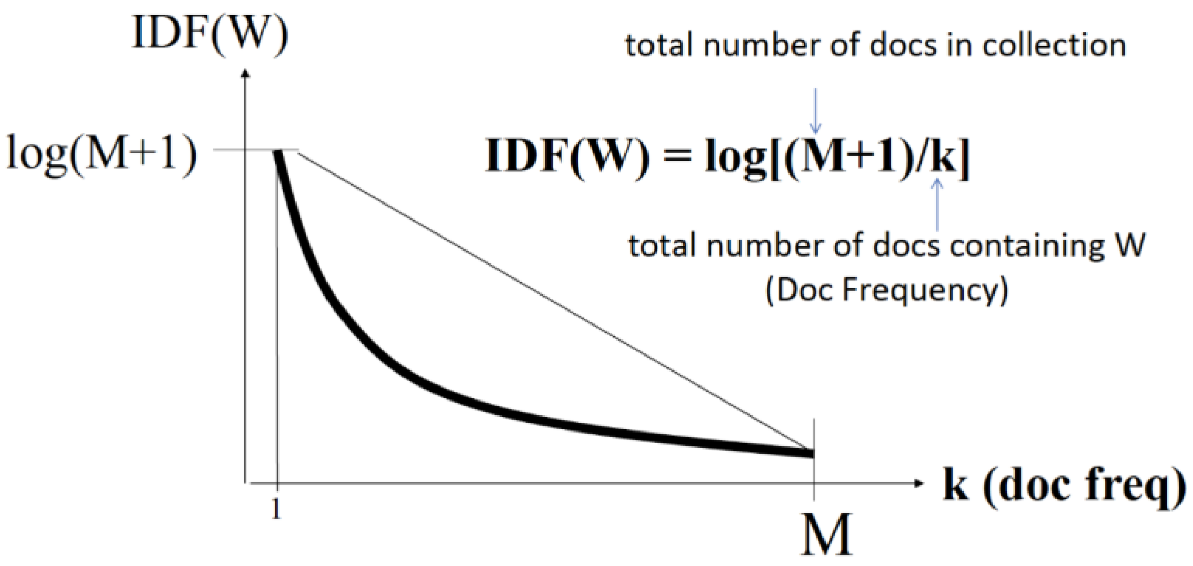
```
x_docFreq=[]
y_IDF=[]
# y_
# for word,docFreq in docFreqofWordDict.items():

x_docFreq=list(docFreqofWordDict.values())
y_IDF=list(IDFdict.values())

# print(x_docFreq)
# print(y_IDF)

len(y_IDF)

plt.plot(x_docFreq,y_IDF,'r.')
```

Out[17]:

```
[<matplotlib.lines.Line2D at 0x2068b9a0ac8>]
```



### sort vocabulary according to IDF-against K (number of documents containing that word) values and plot using matplotlib.pyplot

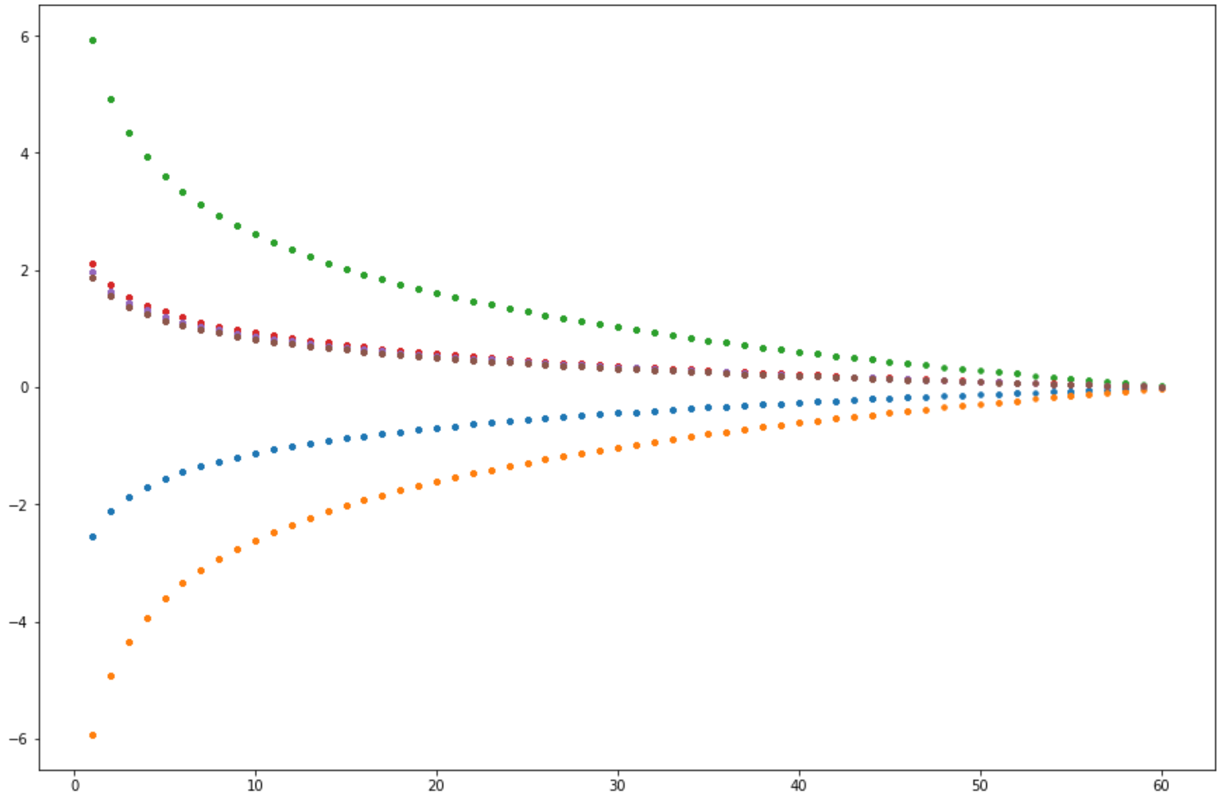

IDF(W)
total number of docs in collection
log(M+1)
IDF(W) = log[(M+1)/k]
total number of docs containing W
(Doc Frequency)
k (doc freq)
1
M

In [28]:

```
logbases=[0.2,0.5,2,3,4,5,6,7,8,9,10]
totalNumOfDocs=len(TextFilesPaths)

inverseDocFreqzerotwo=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDoc
inverseDocFreqzerofive=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDo
inverseDocFreq2=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,2))
inverseDocFreq3=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,3))
inverseDocFreq4=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,4))
inverseDocFreq5=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,5))
inverseDocFreq6=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,6))
inverseDocFreq7=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,7))
inverseDocFreq8=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,8))
inverseDocFreq9=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,9))
inverseDocFreq10=(InverseDocFreqlogbase_func(docFreqofWordDict,totalNumOfDocs,10)



inverseDocFreqzerotwo=list(inverseDocFreqzerotwo.values())
inverseDocFreqzerofive=list(inverseDocFreqzerofive.values())
inverseDocFreq2=list(inverseDocFreq2.values())
inverseDocFreq7=list(inverseDocFreq7.values())
inverseDocFreq8=list(inverseDocFreq8.values())
inverseDocFreq9=list(inverseDocFreq9.values())




x_docFreq=list(docFreqofWordDict.values())

len(docFreqofWordDict)
len(inverseDocFreqzerotwo)


plt.figure(figsize=(15,10))
plt.plot(x_docFreq,inverseDocFreqzerotwo,'.')
plt.plot(x_docFreq,inverseDocFreqzerofive,'.')
plt.plot(x_docFreq,inverseDocFreq2,'.')
plt.plot(x_docFreq,inverseDocFreq7,'.')
plt.plot(x_docFreq,inverseDocFreq8,'.')
plt.plot(x_docFreq,inverseDocFreq9,'.')
```

Out[28]:

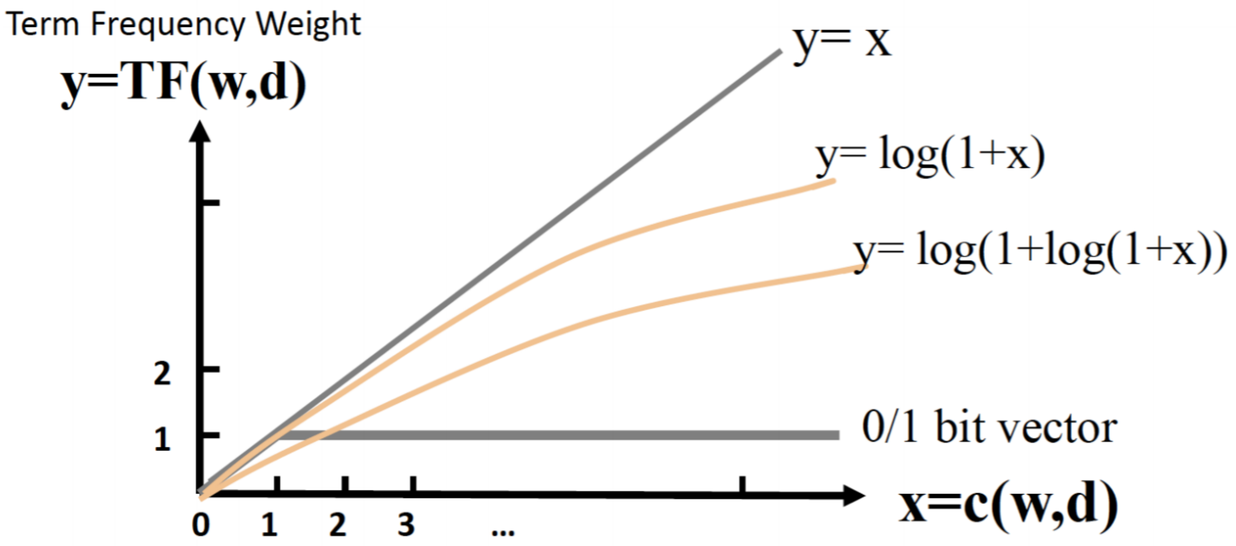
```
[<matplotlib.lines.Line2D at 0x2068b998a20>]
```



### construct a plot that shows how IDF-K relation changes as base of logarithm changes from 10 to -1.

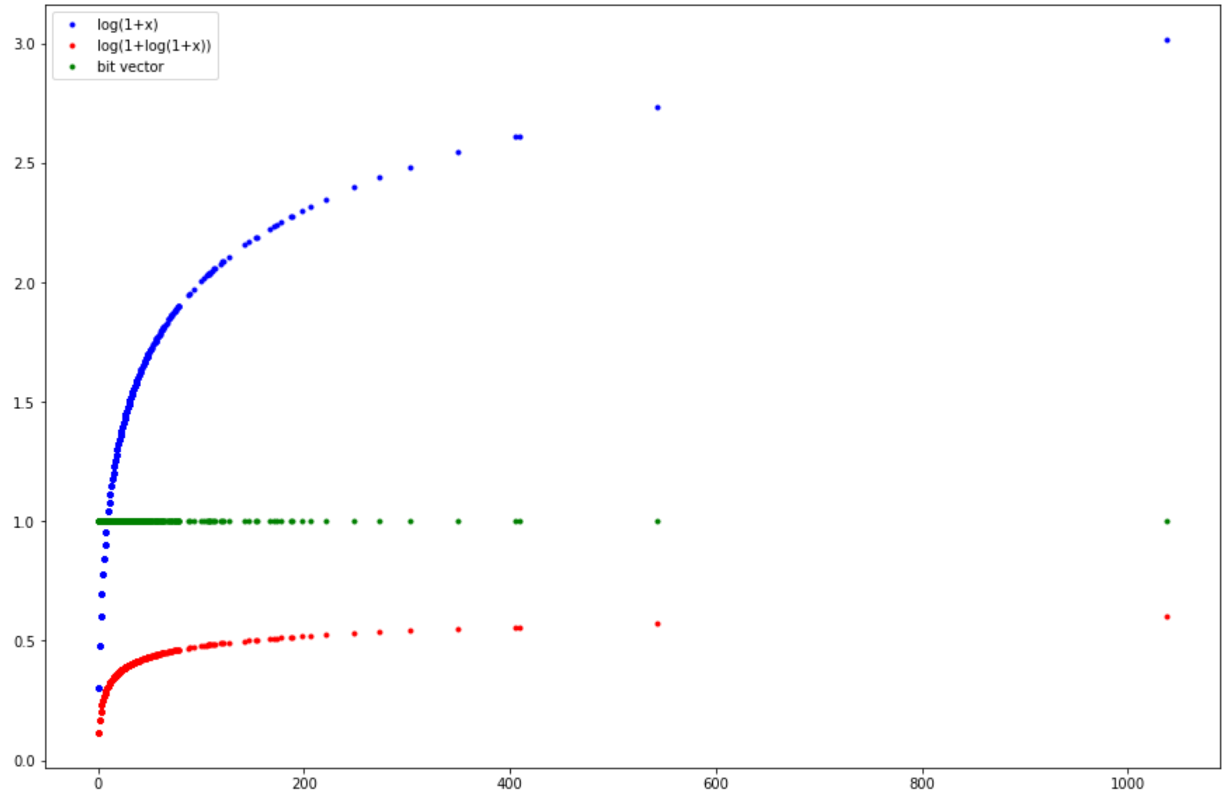
In [ ]:

### construct a plot Term Frequency weight transformation such as this one

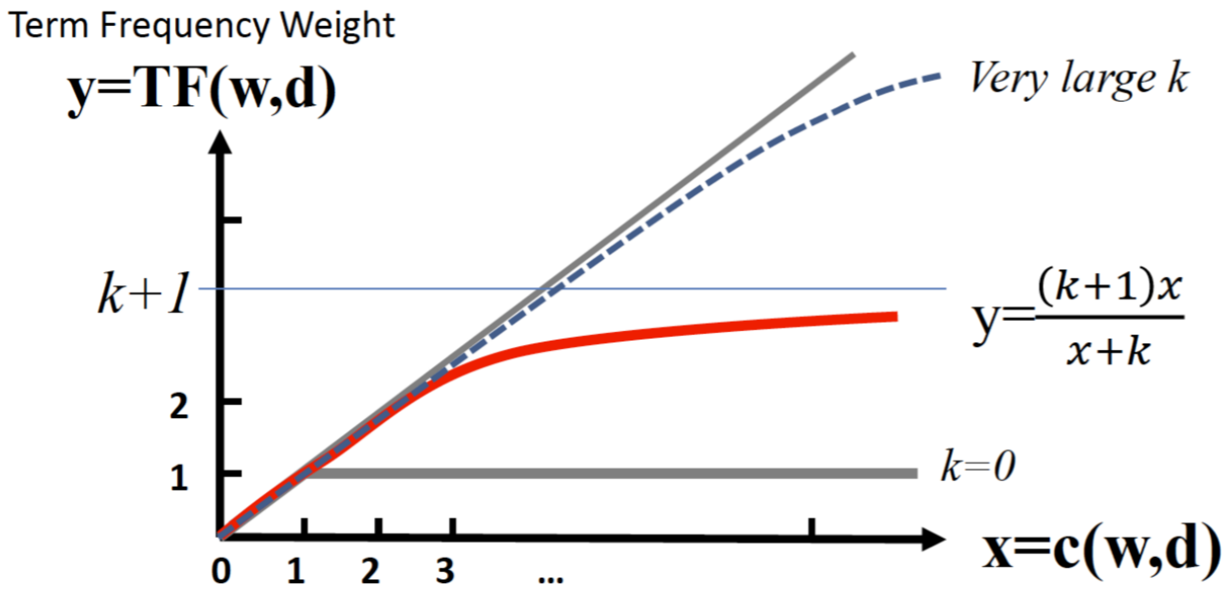


In [19]:

```
import math
countOfWords=[]
templist=[]
for doc,termFreqDict in docDictionary.items():
    templist=list(termFreqDict.values())
    countOfWords.extend(templist)



log10list=[math.log10(count+1) for count in countOfWords]
logLog10list=[math.log10(math.log10(count+1)+1) for count in countOfWords]
bitList=[1 for count in countOfWords]


plt.figure(figsize=(15,10))
plt.plot(countOfWords,log10list,'b.',label= 'log(1+x)')
plt.plot(countOfWords,logLog10list,'r.',label='log(1+log(1+x))')
plt.plot(countOfWords,bitList,'g.',label='bit vector')
plt.legend(loc='upper left')

plt.show()
```

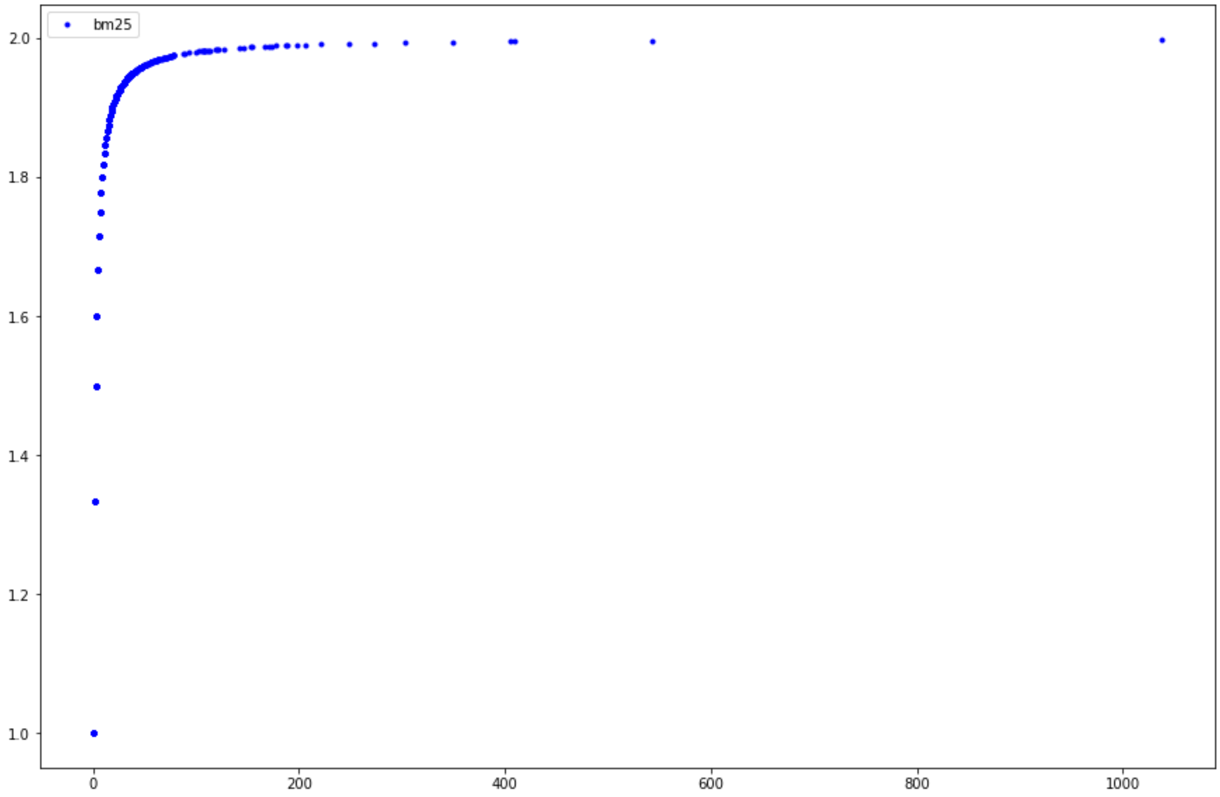


### construct plot of BM25 as shown here



In [20]:

```
k=1
bm25list=[((k+1)*count)/(count+k) for count in countOfWords]

plt.figure(figsize=(15,10))
plt.plot(countOfWords,bm25list,'b.',label= 'bm25')
plt.legend(loc='upper left')

plt.show()
```



In [29]:

```
def pivLengthNormalizer(docList):
    docsLengths = []
    for i in range(0,len(docList)):
        file = open(docList[i],mode='r')
        completeText = file.read()
        file.close()
        wordlistofdoc = wordList(completeText)
        cleanwordlistofdoc = removePuncs(wordlistofdoc)
        filteredSentences = removeStopWords(cleanwordlistofdoc)
        docsLengths.append(len(filteredSentences))
    averageDocLength = sum(docsLengths) / float(len(docsLengths))
    return averageDocLength
```

In [31]:

```
avdl= pivLengthNormalizer(TextFilesPaths)
```

In [32]:

```
import math
```

In [35]:

```
weightedTF={}
newDocDict={}
for docpath,TFdict in docDictionary.items():
    for word,termFreq in TFdict.items():
        x=(math.log10(1+math.log10(termFreq+1)))/avdl
        x=x*IDFdict[word]
        weightedTF[word]=x
    newDocDict[docpath]=weightedTF
```

In [ ]: