

Natural Language Processing
Prof. Amrith Krishna
Department of Computer science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
Tutorial

Hello everyone. Welcome to the first tutorial session for the Natural Language Processing course. In this tutorial session we will be seeing how to set up the programming environment for the course and how to use it to solve your assignment problems. So, we will be using Python as our default programming language and we will be using Jupyter as our default programming environment.

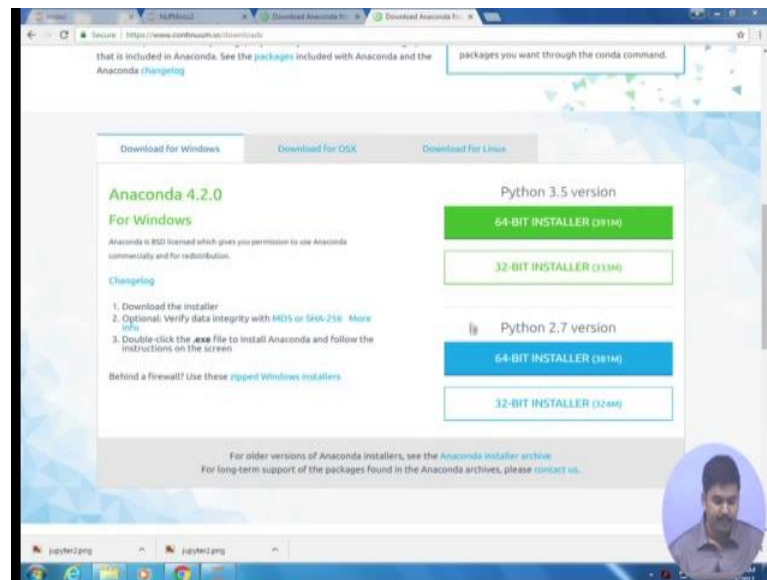
So, if you are already comfortable with a particular programming language or an environment it is perfectly fine that you use one of those, but if you feel that you will be requiring our assistance it is recommended that you use Python and Jupyter.

(Refer Slide Time: 01:16)



So, let us see how to install a programming environment. So, what I will suggest is that you can go to Google and type Anaconda Python. So, Anaconda is a package that gives you the programming language, programming environment and all other necessary packages required to run your course. So, you can see that the first link here which is the Anaconda package leads you to the download page. So, the (Refer Time: 01:46) we get here we can go to the menu and see the download link is up here.

(Refer Slide Time: 01:58)



For installation we can look into the packages here. As we can see the package is available for different operating systems and we recommend that you go with the Python 2.7 version.

So, depending on your OS you can install the 60 bit installer or the 323 bit installer. So, if you are working on OSX or Linux where installers are available here we recommend that you install that (Refer Time: 02:30) version and go to the particular director where the package is downloaded and run your dash command to execute this particular package. The command line installer will take you through and it show you how to install the package.

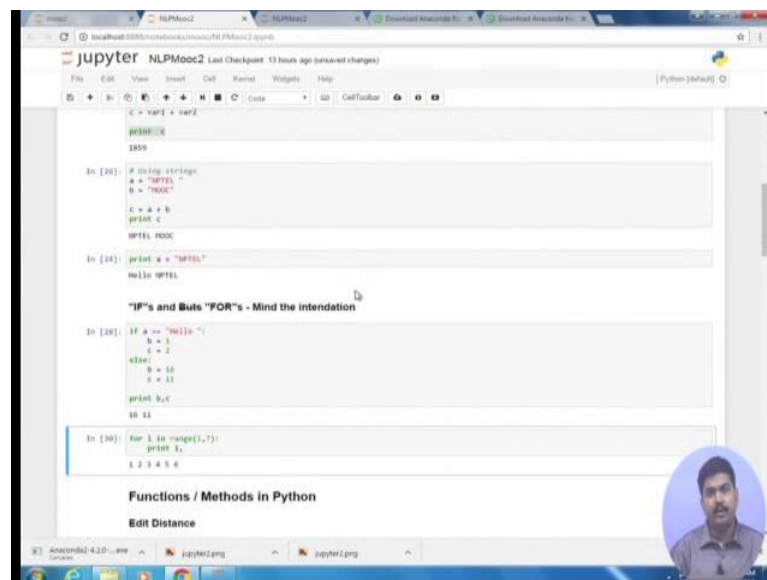
For the windows the graphical installer is available. And you can download it by clicking on this particular link. As I have already have the install version available with me I will be showing you a step by step process of how to install this set up. So, if you have space constraint you can cry the minimal package called as Miniconda otherwise it is advisable that you go with the current installation which is Anaconda.

So, here is the installation package and you can go to a step by step process. And yes, it ask for two separate ways of installation if you do not have admin privileges in the particular system you are installing you should be selecting this one just for your particular use. And you can define why you are package should be install and make sure you have at least 2 GB of free space available in your system. Since just that I have

already a package install. So, I will be just showing you how to install it to on a separate folder and I just you have to make sure this are ticked and click on the install button.

So, the install will take somewhere around 15 to 20 minutes to complete. And once its complete you will be able to see the Anaconda package in your programs and once you click here we can see different packages installed where you should be selecting Jupyter notebook. So, Jupyter notebook is a programming environment that runs from your browser. So, it has a server running at the back. And I have an instance of Jupyter already running in my browser.

(Refer Slide Time: 05:05)



So, it go with by default goes to your document folder and within the document folder it will show you where exactly your files are residing. So, as part of the assignment we will be providing you some help of files of which this is the particular programming file that you should be looking for. So, this is your programming environment.

So, notebook is not just a programming environment it helps you keeps lot of things that you learn, it helps you document the stuff that you basically go through so you can refer it back. So, I will be using Python as my language inside Jupyter. And Python also tends to be my favorite calculator. For example, if you want to calculate just some values I can just see how multiplication of two values this may an answer. So, 987 into 543 will give me this particular answer. Now, if I want to stop using my variables I can gives the values to the variables there is not type declaration as required, we can just give a

variables name so maybe I can give variable 1 and this will be variable 2 and c equal to variables 1 plus variable 2; and then I print the value for c.

Now, assuming since we will be we are studying about natural language processing we will be taking a lot of things and lot of textual data to handle. We how to stop text strings. So, if you have variables you can just assume text strings enclosed with in quotes. So, here I have typed 'hello' and world and I do the plus operation to both the variables let see what happens. So, the particular variables strings hello world together which means it concatenation in a both the variable. So, this automatic identification whether the sum needs to be done or concatenation needs to be done this handled by the Python interpreter.

Suppose if I change and I say 'hello mook' let us say how with becomes, yes; so the output given here basically shows 'hello mook'. Similarly, a variable initialized at a particular cell can be reused at another cell as it resides in your memory. For example, I reused the 'hello' which was already defined in the particular cell here and I use hello in NPTEL let see how it works.

And important thing that you have to care of is indentation. So, when using conditional statements like the 'if' or the loops like for loop you have to make sure that the block indentation is maintained. So, what I want to check is that if 'a' is equal to hello. So, I check for whether the variable 'a' contains the variable hello then the variables are given the values 1 and 2 otherwise the variables b and c are given the values 10 and 11. Let see which value gets print.

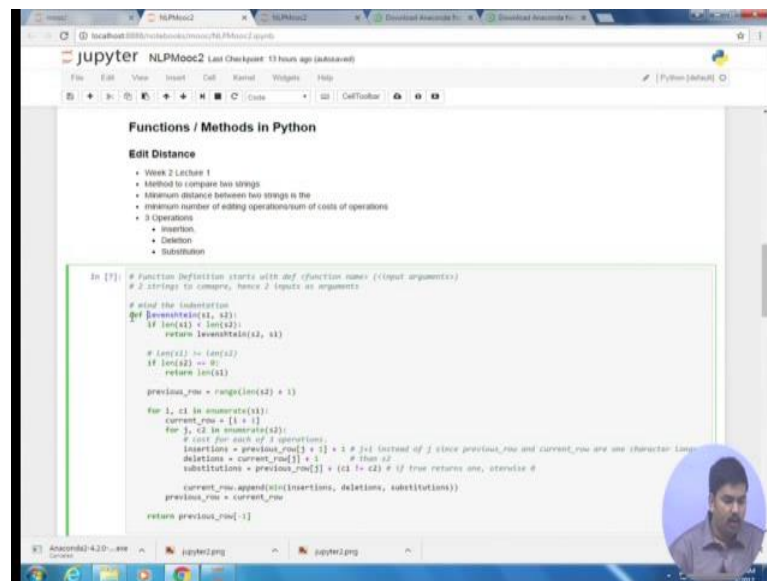
Since the variable a has the value hello it prints 1 and 2. Suppose I change it to NPTEL and execute it and now I run here. So, here I get the values 10 and 11. So, if you are still confused how to run the code after typing the code in the particular cell we can either use the shortcut control and enter or you may go to this button and this will execute the code. So, let us change the variables values here and see if what happens if they enter code. Now b gets the values 60 and that gets printed here.

Now, suppose I am using a loop to print all the values between 1 and 5 that is 1 to 4 how do I do. So, I defined keyword 4, I define a variable and I define that the variable will be within a range from 1 to 5. If we can make it 7 or any value of your choice and let us see.

So, it prints all the values from 1 to 6. If I do not want to print it line by line I can just use a comma here and it gives me the values written in the same line.

So, this is some basic syntax that will let you coding Python and we recommend that you go through some of the basic tutorials regarding Python and some of the basic data structures that Python has so that it will help in making coding experience much better.

(Refer Slide Time: 10:57)



```
In [7]: # Function definition starts with def (function name) (input arguments)
# 2 strings to compare, hence 2 inputs as arguments

def levenshtein(x1, x2):
    if len(x1) < len(x2):
        return levenshtein(x2, x1)

    # len(x1) >= len(x2)
    if len(x2) == 0:
        return len(x1)

    previous_row = range(len(x2) + 1)
    for i, c1 in enumerate(x1):
        current_row = [i + 1]
        for j, c2 in enumerate(x2):
            # cost for each of 3 operations
            insertions = previous_row[j + 1] + 1 # i+1 (instead of j) since previous_row and current_row are one character longer
            deletions = current_row[j] + 1 # then x2
            substitutions = previous_row[j + 1] + (c1 != c2) # if true returns one, otherwise 0
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]
```

In order to provide that we will be providing you some links and some resources that you can go through and get fresh up with the particular code. So, as part of your assignment you are supposed to solve questions that involved edit distance that involves calculating the likelihood using bigrams and unigrams and I will be showing you how to handle this.

As from the week two lecture one you might be aware of the concept called as Edit Distance. So, edit distance is a particular function that helps you compare two strings and find the similarity between them. So, the edit distance gives you the minimum distance between two strings and the minimum distance can be calculated in terms of the number of operations that required to change one string from to another. And this is defined in terms of three operations insertion, deletion and substitution.

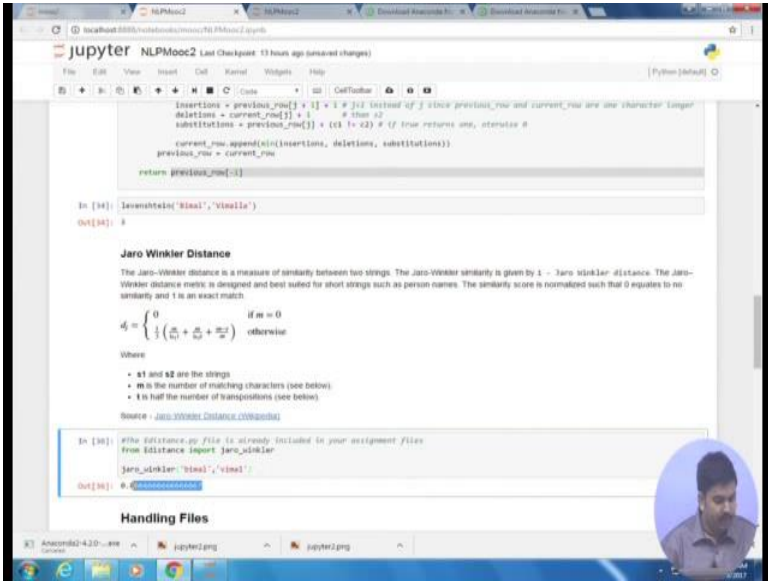
So, we have provided a function called Levenshtein that helps you execute or calculate the edit distance. So, this is how functions are methods are used in Python we use the keyword def followed by the name of the function. And since I have two strings to come

I give the strings as my function arguments that, so that when you call the function you have to give which are the two strings to be given.

Now, these are the three operations; the insertion, deletion, substitution that you will be using to compare both the strings. So, we need not worry about what exactly is happening inside you can think of it as a black box and you can just execute it. But if you are curious, you can obviously work around with it hack around with it and see how the output changes.

So, assuming I already have the code setup and I already executed this particular cell so that the function goes to gets invoke. Now I call this function Levenshtein and I give two strings which is h e l l p and h e l l o and I want to see what will be the difference between the strings or what will be the edit operation edit distance. It turns out that the edit distance is 1 which is being written by this particular variable here.

(Refer Slide Time: 13:40)



The screenshot shows a Jupyter Notebook interface with a code cell containing a Levenshtein distance function. The function takes two strings, 's1' and 's2', and returns the edit distance. Below the code cell, the function is called with 'h e l l p' and 'h e l l o' as arguments, and the output is 1. Below the code cell, there is a section titled 'Jaro Winkler Distance' which explains the metric and provides a formula. The formula is:
$$J_s = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{2} \left(\frac{a_1}{n_1} + \frac{a_2}{n_2} + \frac{a_3}{n_3} \right) & \text{otherwise} \end{cases}$$
 where a_1 and a_2 are the strings, m is the number of matching characters, and t is half the number of transpositions. Below the formula, there is a source link: [Source - Jaro-Winkler Distance - Wikipedia](https://en.wikipedia.org/wiki/Jaro-Winkler_distance). At the bottom of the notebook, there is a section titled 'Handling Files'.

```
def levenshtein(s1, s2):
    if len(s1) < len(s2):
        s1, s2 = s2, s1
    previous_row = None
    current_row = []
    for i in range(1, len(s2) + 1):
        insertions = previous_row[-1] + 1 if i > len(s1) else previous_row[i]
        deletions = current_row[i] + 1 if i > len(s1) else current_row[i]
        substitutions = previous_row[i] + (1 if s1[i-1] != s2[i-1] else 0)
        current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]
```

```
In [34]: levenshtein("h e l l p", "h e l l o")
Out[34]: 1
```

Jaro Winkler Distance

The Jaro-Winkler distance is a measure of similarity between two strings. The Jaro-Winkler similarity is given by $1 - \text{Jaro-Winkler distance}$. The Jaro-Winkler distance metric is designed and best suited for short strings such as person names. The similarity score is normalized such that 0 equates to no similarity and 1 is an exact match.

$$J_s = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{2} \left(\frac{a_1}{n_1} + \frac{a_2}{n_2} + \frac{a_3}{n_3} \right) & \text{otherwise} \end{cases}$$

Where:

- a_1 and a_2 are the strings
- m is the number of matching characters (see below)
- t is half the number of transpositions (see below)

Source - [Jaro-Winkler Distance - Wikipedia](https://en.wikipedia.org/wiki/Jaro-Winkler_distance)

```
In [36]: with Editance.py file is already included in your assignment files
from Editance import jaro_winkler
jaro_winkler("h e l l p", "h e l l o")
Out[36]: 0.8571428571428571
```

Handling Files

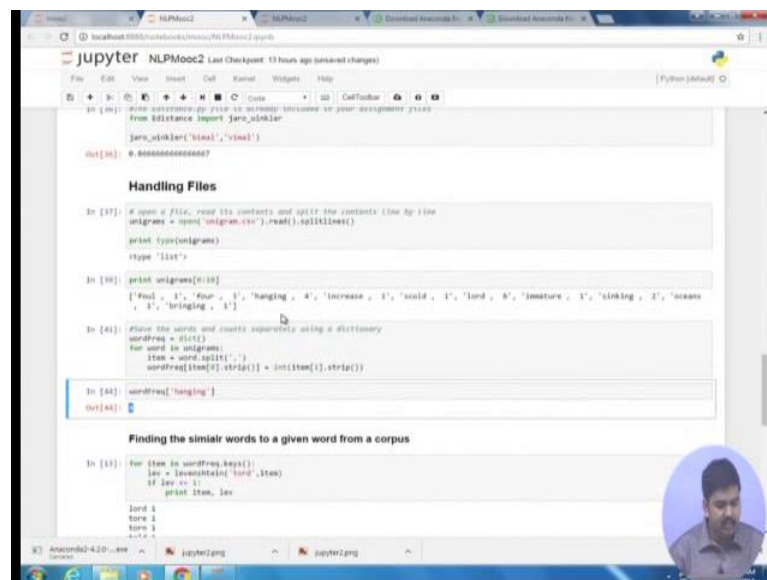
Suppose I want to give; yes it also gives the same values. So, if I have a new string the value changes. As part of your assignment you have one question where you are supposed to solve the problems using Jaro-Winkler distance. Jaro-Winkler distance is a variant of edit distance that is used to combine strings, and we have provided the explanation of how to calculate the Jaro-Winkler distance between two strings. And you can read about more in the Wikipedia link provided that.

For your convenience I have we have already implemented the Jaro-Winkler as a function and we have stored it in the packages edit distance. So, we can call the Jaro-Winkler function and you can again assume it to be black box just to solve your assignment. So, in Jaro-Winkler as you can see the value ranges between 0 to 1. And if two strings are exactly the same they will provide you the value of 1 and if they are complete different they will provide a value of 0. So, if I give the same string I get a value of 1 as my Jaro-Winkler distance.

So, this is very useful when it comes to combine strings such as person names because there are each variations in the person name that we use. For example, it is very common that the word Vimal might be spelled as Bimal depending on your demography. So, suppose we apply this function and let see how the string gets the value. So, the similarity between Bimal and Vimal turns out to be 0.867.

Now let us see given a data set or a corpus of words and you want to find the most similar words of a given string from that how do we do that.

(Refer Slide Time: 15:37)



```
from editdistance import jaro_winkler
jaro_winkler('vimal','vimal')
Out[36]: 0.8666666666666667

Handling Files
In [37]: # open a file, read its contents and split the contents line by line
unigram = open('unigram.csv').read().splitlines()
print type(unigram)
<type 'list'>

In [38]: print unigram[0:10]
['fuel', '1', 'four', '1', 'hanging', '4', 'increase', '1', 'solid', '1', 'lord', '6', 'mature', '1', 'sinking', '2', 'ocean', '1', 'bringing', '1']

In [41]: # place the words and counts separately using a dictionary
wordfreq = dict()
for word in unigram:
    item = word.split(',')
    wordfreq[item[0].strip()] = int(item[1].strip())

In [44]: wordfreq['hanging']
Out[44]: 4

Finding the similar words to a given word from a corpus
In [43]: for item in wordfreq.keys():
    lev = levenshtein('lord',item)
    if lev <= 2:
        print item, lev
lord 1
four 1
two 1
one 1
```

So, you will be provided with a file called unigram dot csv, you need to load that file and you have to. So, every line contains a single word or every unique word and the frequency with which it appears in a particular purpose.

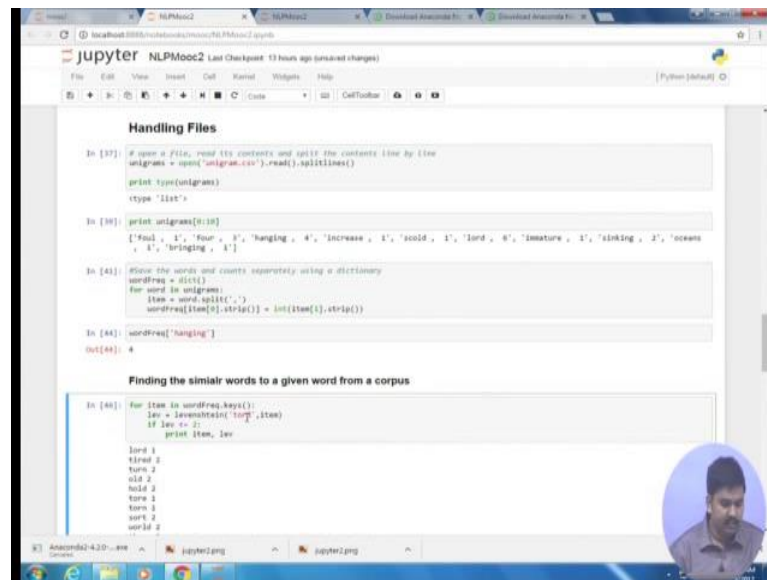
So, I read the file to the variable `unigrams`; and `unigrams` will be store as a list there is a default data structure used in Python you can ignore about list from the Python manuals. So, for convenience you can think of it as an array where if I want to access the first element I just give the index of the first entry which is 0 and I print that well. So, the entry gives me `foul` for which has a frequency of 1. Suppose I want to say the first ten entries in the file I will use the colon followed by number 10, so it will give me all the numbers from 0 to 9 and that gives me the first 10 entries which is `foul` which appears 1 time `foul` which appears 3 times and `immature` appears 1 times `bringing` appears 1 time.

Now, I will be using a different data structure called as dictionary that helps me separate the count and the word from one another. So, dictionary as the name suggest we have a key where the key is your word and when I provide the key to the dictionary it will provide me the count with which it is appearing. So, here is the function that helps me go and it stores the value inside the dictionary called as `wordfreq`.

So, once I execute it. So, a dictionary has two entities; one it is called as the keys and it has it made as a list of the keys. So, if you type `wordfreq` dot `keys` it will provide you all the keys that it has. Now if I want to know how many times `thunder` is appearing, what I would do is I would see the count of `thunder` by just giving the key as `thunder` to the variable `wordfreq`. Now it appears only once let us see how many times `hanging` is appearing and that is validate with that it is coming out to be correct. It is coming out to be correct as its 4 here and we already have checked it was 4.

Now, since you know how to open a file how to store it in a dictionary and you already also know how to compare two strings, let us see we can whether we can find the most similar word to a given string or a given word.

(Refer Slide Time: 18:46)



```
Handling Files

In [37]: # open a file, read its contents and split the contents line by line
unigrams = open('unigram.csv').read().splitlines()
print type(unigrams)
<type 'list'>

In [38]: print unigrams[0:10]
['foul', 'l', 'four', 'b', 'hanging', '4', 'increase', 'l', 'scold', 'l', 'lord', 'b', 'immature', 'l', 'sinking', 'd', 'oceans', 'l', 'bringing', 'l']

In [41]: # store the words and counts separately using a dictionary
wordfreq = dict()
for word in unigrams:
    wordfreq[word.strip()] = int(wordfreq.get(word.strip(), 0) + 1)

In [44]: wordfreq['hanging']
Out[44]: 4

Finding the similar words to a given word from a corpus

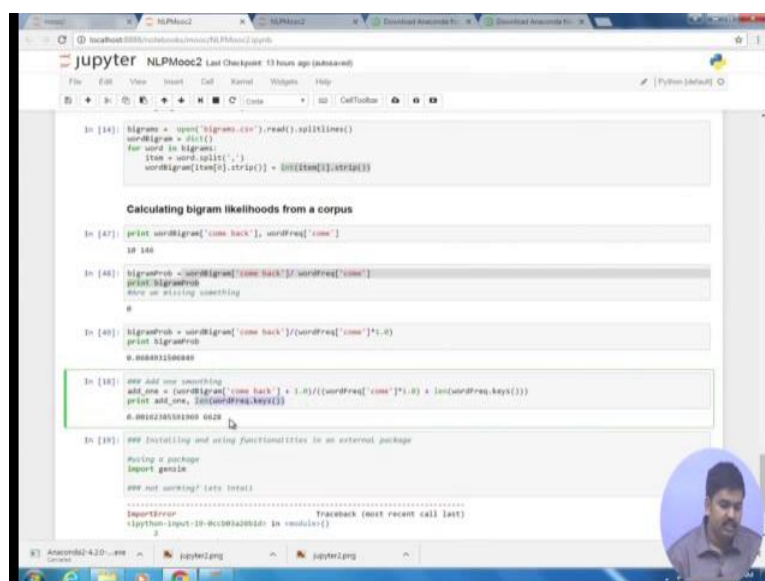
In [46]: for item in wordfreq.keys():
    lev = levenshtein('tord', item)
    if lev <= 2:
        print item, lev
lord 1
tired 2
torn 2
old 2
hold 2
tore 1
torn 1
word 2
world 2
```

So, what I do is that I iterate through all the possible words in the dictionary and then I find the Levenshtein distance between the string that I want to come here which happens to be t o r d. It is a spelling mistake. So, I have to find which word be replaced with t o r d.

And if you see here I take each word from the keys of the dictionary and assign it to the variable item one at a time and I compare tord with item. And I am going to print all the words that has a Levenshtein distance which is less than or equal to 1. So, if you find tord well and good otherwise let us see what are the other words.

Once it turns out there are five words which are similar to t o r d; which is lord tore torn told or word. Let us see all the words which have a Levenshtein distance of less than or equal to 2 when it turns out there are much more entries which have a Levenshtein distance of two with words.

(Refer Slide Time: 20:12)



```
In [34]: bigrams = open('bigrams.csv').read().splitlines()
wordbigram = {}
for word in bigrams:
    item = word.split(',')
    wordbigram[item[0].strip()] = item[1].strip()

Calculating bigram likelihoods from a corpus

In [47]: print wordbigram['come back'], wordfreq['come']
10 140

In [48]: bigramprob = wordbigram['come back']/wordfreq['come']
print bigramprob
0.07142857142857142

In [49]: bigramprob = wordbigram['come back']/(wordfreq['come']**0.5)
print bigramprob
0.004031506000

In [50]: new add one something
add_one = wordbigram['come back'] + 1.0/(wordfreq['come']**0.5) + log(wordfreq.keys())
print add_one, log(wordfreq.keys())
0.0012185181800 0.020

In [51]: new Installing and using functionalities in an external package
Using a package
import pandas

new not working? lets install
Traceback (most recent call last):
  <ipython-input-51-0c3b9a2b0d0> in <module>()
    1
```

Now what will be doing either you are given with another file called the bigrams dot csv, and it contains all the bigrams with the frequency with which the bigrams are appearing.

So, this is very straight forward, just like what we have done previously for unigram dot csv we are going to use the same set of code if you see here. But here I have just written it in 1 lag. So, I have the bigrams file which I have read and I have spitted into a list then I define the dictionary called wordbigram and I store every bigram as my key and the count as the value. So, bigrams and unigrams as told as now two different dictionary; one is in wordfreq and other is in wordbigram.

Now, suppose I want to calculate the bigram likelihood. As we know the formula is the count with which the bigram occurs by the count with which the first word in the bigram occurs. So, if I want to calculate the bigram likelihood of comeback I want in other frequency with which the comeback is occurring in a particular corpus and the frequency with which come is occurring. And it turns out that the value is occurring 10 times and come is occur 140 times. So, if I want to calculate the probability of come back and come I would do wordbirgram come back divided by word frequency of come which will give me 10 by 146.

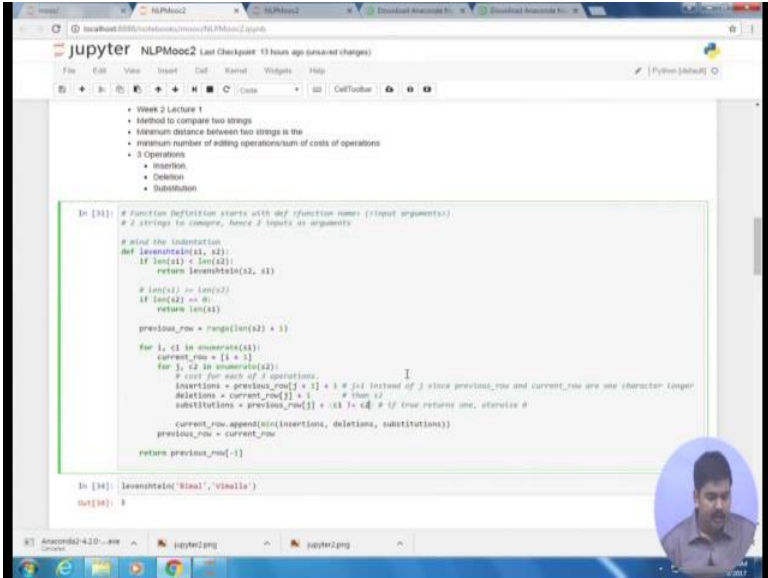
It turns out that the value gives coming out to be 0 so there is certainly some problem. The problems is that since both come the frequencies are individual values the system assumes the answer also should coming an integer, but it turns out that the value is

coming out to be a floating point number. So, to handle up there are different ways of explicit type casting what I am doing here is just multiply the value with a 1.0 so that before the division occurs one of the value turns out to be a float. After the operation is done the value turns out to be 0.068.

Now suppose we want to calculate the likelihood of the same bigram, but using add one smoothing as we know. Add one smoothing basically adds a 1 to the numerator for the bigram likelihood and it also adds to the vocabulary size or the number of unique words in a corpus. To get the number of unique words in a corpus we can just take all the keys in the dictionary and we can find the count of that and the number of keys. And it turns out that the function length performs the same. So, I have used the length function to find all the length of the keys. And with that I print the, add one likelihood after add one smoothing. And we can also see how many unique words are there in the corpus.

So, this is pretty much for what you can do with your assignment 2. And there will be one question where you are supposed to modify the course in word for the particular operation.

(Refer Slide Time: 24:10)



The screenshot shows a Jupyter Notebook interface with a file explorer at the top showing 'Week 2 Lecture 1'. The notebook contains a Python function for calculating the Levenshtein distance between two strings. The function is defined as follows:

```
In [31]: # function definition starts with def (function name) (input arguments)
# 2 strings to compare, hence 2 inputs as arguments

# mind the indentation
def levenshtein(s1, s2):
    if len(s1) < len(s2):
        return levenshtein(s2, s1)

    # len(s1) == len(s2)
    if len(s1) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            # cost for each of 3 operations
            insertions = previous_row[j + 1] + 1 # i instead of j since previous_row and current_row are one character longer
            deletions = current_row[j] + 1 # then i2
            substitutions = previous_row[j] + (1 if c1 != c2 else 0) # if true returns one, otherwise 0
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]
```

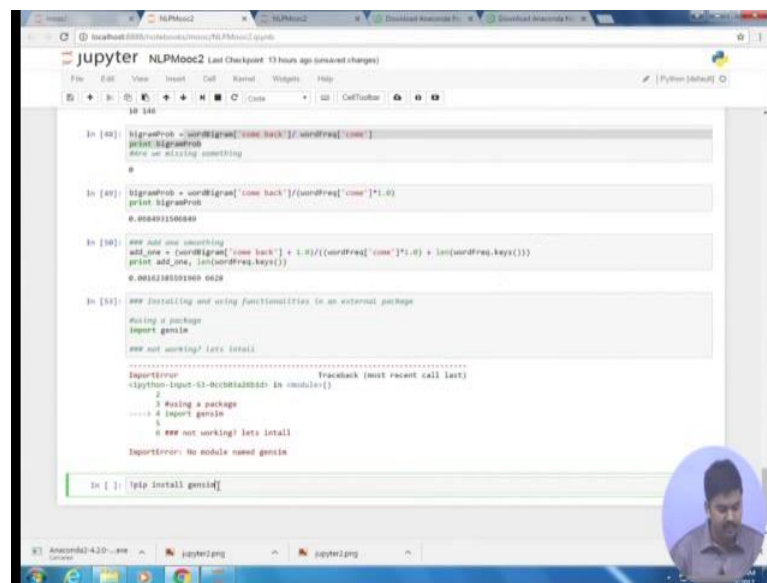
The function is then called with the strings 'kmal' and 'vinalla'.

```
In [34]: levenshtein('kmal', 'vinalla')
Out[34]: 5
```

So, in that particular question you are supposed to change the value for substitution. As insertion and deletion involve at cost of 1, but substitution involves a cost of 2; in order to do that I am going to modify the value.

So, what this particular function does is that it compares whether two characters at a position are same. So, if they were not same it returns a 1 because this is a logical operation otherwise it returns a 0. So, previously the cost was 1, so it was left (Refer Time: 24:54). Now since the cost is two and just modifying it to be two here. And let us see how much does it cost and how Bimal and Vimal compare with this (Refer Time: 25:05) operation. And it turns out that the value becomes 2. This is generally done when to want to penalize substitutions and you want to check more into the insertions and deletions operations.

(Refer Slide Time: 25:26)

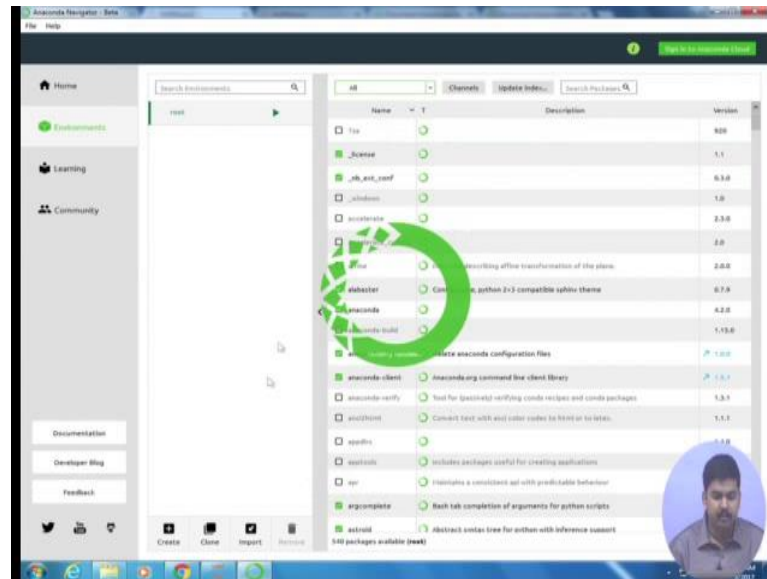


```
In [43]: sig_freq = word_freq['come back'] if word_freq['come']:  
print sig_freq  
else:  
    we are missing something  
0  
  
In [44]: sig_freq = word_freq['come back']/(word_freq['come']**0.5)  
print sig_freq  
0.008401150089  
  
In [45]: we add one something  
add_one = word_freq['come back'] + 1.0/(word_freq['come']**0.5) + len(word_freq.keys())  
print add_one, len(word_freq.keys())  
0.0016218519189 6428  
  
In [46]: we installing and using functionalities in an external package  
Using a package  
import gensim  
  
we not working! lets install  
.....  
ImportError: Traceback (most recent call last)  
2  
3 Using a package  
4 import gensim  
5  
6 we not working! lets install  
ImportError: No module named gensim  
  
In [ ]: !pip install gensim
```

And for some aspects you might require some external packages to be installed and I will be showing you how to install; an external packages. Suppose I know there exist a package gensim and I assume that it is installed in this system. I will invoke that external package using import gensim. But, it turns out that the package is not installed. So, Anaconda comes with a default package manager which you can select using here Anaconda navigator. So, Anaconda navigator provides you all the different packages and it provides a repository way of different packages that you can install.

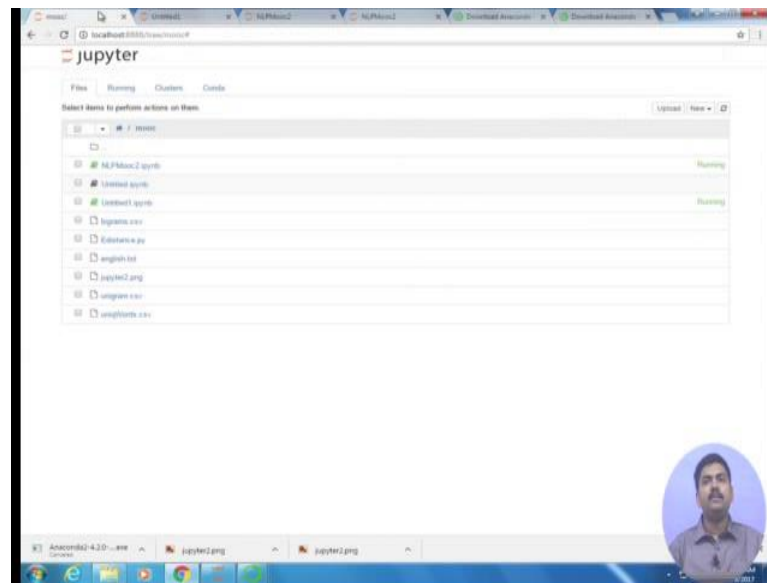
Alternatively, you can create a new cell and you can install a package within the notebook itself. What you have to do is that you have to mind this exclamation mark so you have type pip install gensim and this will also help you install the particular package.

(Refer Slide Time: 27:27)



So, here you can click on to the environment tab and you can see for all packages are reliable in the repository of Anaconda and you can just type gensim and you can click here you can say electric and you can apply it to install. As I was suggested you may alternatively use this command to install.

(Refer Slide Time: 27:56)



If you want to create a new notebook you should click on the new and you should select one of this. So, this will give me a new notebook where you can have a fresh notebook, where you can try out code by yourself. So, this is pretty much.

Thank you.