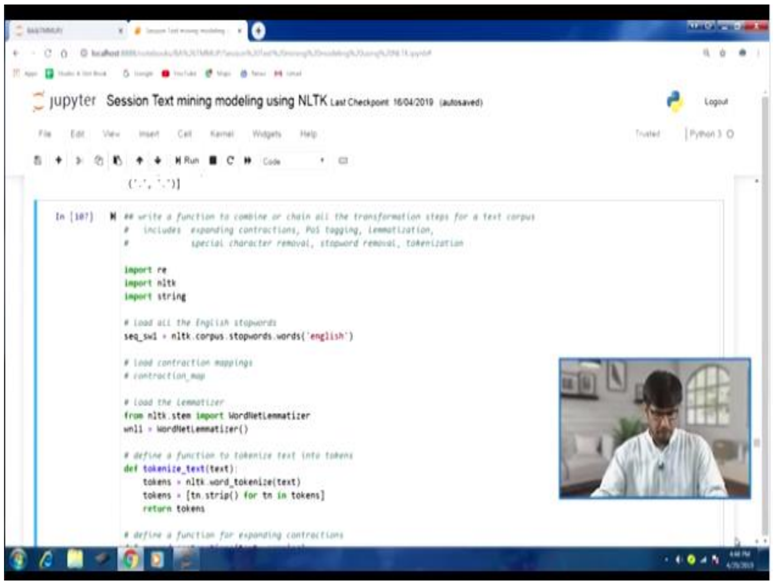
Business Analytics And Text Mining Modeling Using Python

Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-40
Text Mining And Modeling-Part III

Welcome to the course business analytics and text mining modeling using python, so in previous lecture we talked about a number of steps related to you know transforming unstructured text, stemming, lemmatization and many other aspects as well. Now we will continue our discussion on some of these transformation steps, rather will like to combine whatever we have discussed till now. So, all the transformation steps that we have been able to cover in previous few lectures, we like to combine them in you know in one function.

(Refer Slide Time: 01:03)


jupyter Session Text mining modeling using NLTK Last Checkpoint 16/04/2019 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Logout
Trusted
Python 3

```
('.', '.')]
```

```
In [107]: ## write a function to combine or chain all the transformation steps for a text corpus
          #   includes  expanding contractions, POS tagging, lemmatization,
          #             special character removal, stopword removal, tokenization

          import re
          import nltk
          import string

          # load all the English stopwords
          seq_sw1 = nltk.corpus.stopwords.words('english')

          # load contraction mappings
          # contraction_map

          # load the lemmatizer
          from nltk.stem import WordNetLemmatizer
          wnl1 = WordNetLemmatizer()

          # define a function to tokenize text into tokens
          def tokenize_text(text):
              tokens = nltk.word_tokenize(text)
              tokens = [tn.strip() for tn in tokens]
              return tokens

          # define a function for expanding contractions
```

So, let us do that, so specifically will be combining this expanding contractions, POS tagging, lemmatization, special character removal, stop word removal and tokenization. So, for this these are some of the you know modules that we might require re nltk string and you can see next thing is load all the English stop words. So, this will be referring to, then contraction mapping, contraction map we have already you know define.

So, that will also be used then lemmatizer this world net one at lambda lemmatizer one instance of this call WNL 1, you would be using here.

(Video Starts: 01:42)

Then next is you know a tokenized you know text this function we used in the previous lecture also. So, nothing is special in this it is just a wrapper for this our nltk.word_tokenize and for you know this is being called for a list of tokens in a sense. So, you can see a list comprehension there as well in the definition, so this is our tokenize_text functions. We can pass on it a text, we can tokenize.

And then a list comprehension to strip off you know some of the you know wide spaces and other things. So, then is defining function would expanding contraction, so this is very much similar to what we have discussed in the previous lecture, so this part is same just in a functional form here. And define a function for lament lemmatization, so here you can see few changes in the sense that we have defined a function you know POS_tag_text, within this you know another function is defined that is essentially to convert penn treeback tag to bag to world net tag.

Because you know that is deeply require for you know because worldnet lemmatizer that class lemmatization that we have to perform, it actually checks for you know pure stag annotation based on wordnet formats. So, we will have to convert the you know penn treebank text to wordnet you know text. So, for this we have defined this function, so essentially it is doing a mapping sort of thing, so any POS tag that is starting with capital J will WN.ADJ and similarly for others also.

Now next line there we are tokenizing the text and then these you know using these tokens we are attaching US tax there using another nltk.POS_tag. And then next thing is we are doing you know calling this you know this penn_WN_tags function within a you know list comprehension here again. So, for word come appears tag in tokens_t we are you know one thing we are converting this you know penn treebank base tag to wordnet tag.

And we are also doing a case you know transformation here word.lower, so in this function POS_tag text. Finally we will have the kind of POS tagging that could be useful in the limit lemmatization, so next function is lemmatize__text. So, in this you can see here we have you know first thing we have POS tagging we are calling that function that we just discussed POS_tag_text. And then next thing we are lemmatizing the text we again here we are using list comprehension.

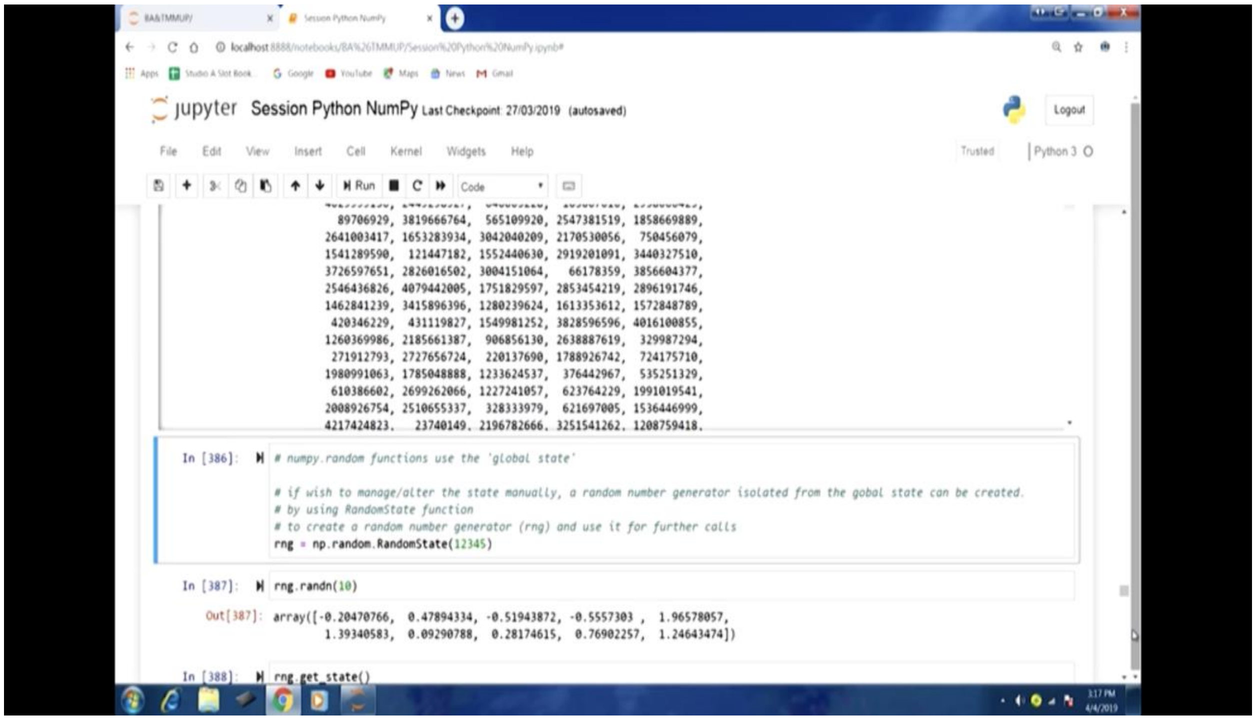
So, for word and,POS tag and tokens P you know we are performing processing this lemmatization, we are calling this you know lemmatize function here. Word, POS_tag because as you would remember that you know we have to specify the part of a speech along with the you know word there. So, lemmatization will be performed here then we concatenate you know then we concatenate all these lemmatized tokens here.

So, this will complete our you know this lemmatize_text function here. Then we have another function remove_special characters, so this is again the same code you know has put into a you know function form. So, you can see, so this will remove all these special characters you know that might be present in the text. Then we have remove stop words, so this part also we discuss in the previous lecture.

So, for this we are using the you know this function here and it will perform that stop word remover, the code is similar to what we discuss in the previous lectures. Now this is the next function is the most important function this is the transform_corpus function. So, all these steps that we have discuss till now, now they are going to be call the related function associated function with those steps are going to be called within this function.

So, in a sense we are combining or chaining all the transformation steps in this function, so that is why we are calling this transform_corpus. So, will be passing on corpus here, so corpus as we talked about it is collection of text documents. So, we are expecting a number of text documents here, so you can see corpus_t and it is empty list here. So, list of you know text document which is essentially we can consider you know sentences or paragraphs of text in a list you know kind of form.

(Refer Slide Time: 03:36)



```
       89706929, 3819666764,  565109920, 2547381519, 1858669889,
     2641003417, 1653283934, 3042040209, 2170530056,  750456079,
     1541289590,  121447182, 1552440630, 2919201091, 3440327510,
     3726597651, 2826016502, 3004151064,   66178359, 3856604377,
     2546436826, 4079442005, 1751829597, 2853454219, 2896191746,
     1462841239, 3415896396, 1280239624, 1613353612, 1572848789,
      420346229,  431119827, 1549981252, 3828596596, 4016100855,
     1260369986, 2185661387,  906856130, 2638887619,  329987294,
      271912793, 2727656724,  220137690, 1788926742,  724175710,
     1980991063, 1785048888, 1233624537,  376442967,  535251329,
      610386602, 2699262066, 1227241057,  623764229, 1991019541,
     2008926754, 2510655337,  328333979,  621697005, 1536446999,
     4217424823.   23740149. 2196782666. 3251541262. 1208759418.

In [386]:  # numpy.random functions use the 'global state'

           # if wish to manage/alter the state manually, a random number generator isolated from the gobal state can be created.
           # by using RandomState function
           # to create a random number generator (rng) and use it for further calls
           rng = np.random.RandomState(12345)

In [387]:  rng.randn(10)

Out[387]: array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.96578057,
                  1.39340583,  0.09290788,  0.28174615,  0.76902257,  1.24643474])

In [388]:  rng.get_state()
```

So, for text in corpus will perform these you know steps, so first expand contraction function is being called and then lemmatize_text is being call then remove special characters then remove stop words. So, all these you know functions are being called here and all the transformation steps in a sense are you know combined here. Then we once all these processing is done, then the corpus_t the empty corpus that we just defined will be appending this text there.

And in this fashion will be doing all the processing for a particular text will take it from the corpus run a group and you know append thus processed or transform text in our new corpus list there. So, this function will have written as the transformed corpus alright, so all the transformation steps would actually be you know performed within function. So, this we have already find let me run again.

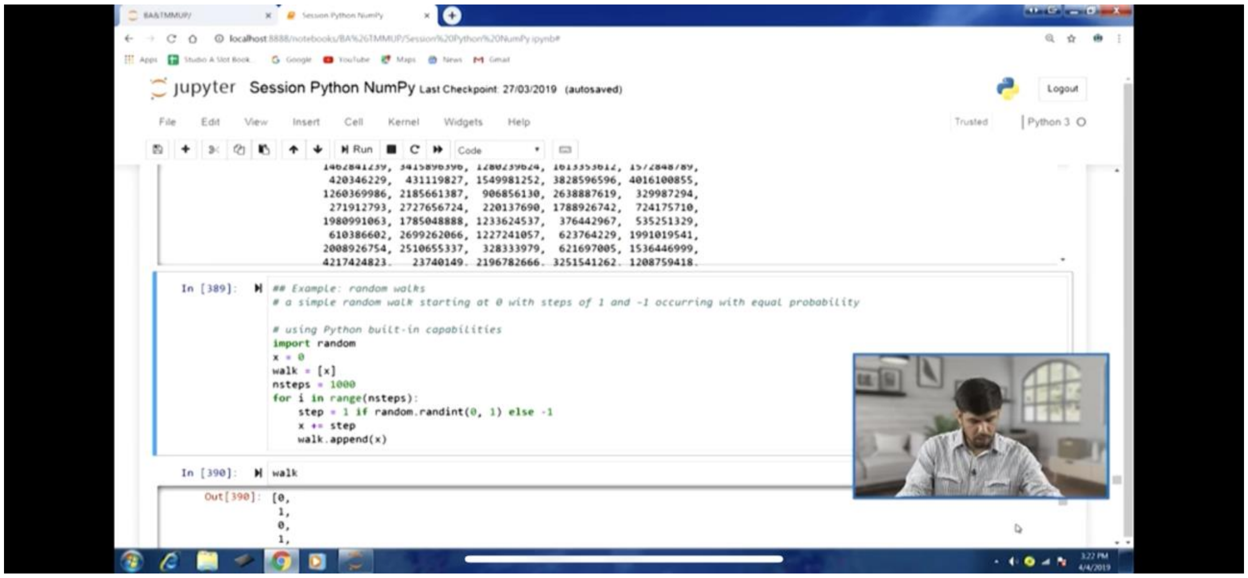
So what will do now will apply this function transform_corpus on emma_sense. So, we will use transform_corpus function to transform or normalize the text corpus here. So, emma_sense we have you know we have you know created before, so let me call this here and it will take some time to complete actually. Because right now emma_ you know this is based on just 1 text file.

And we have you know we had then the sentence tokenization and we had got this. So, right now you know even this took this much time. If we apply later on will be applying transform_corpus to a you know larger corpus rather a proper corpus and you will see it will take slightly more time there. So, let us have a look at this transformed corpus. Now let us have a look at the text you can see just by looking at the text.

You can immediately see that this has been processed quite well, emma, Jane, Austen, 1816, volume, chapter you can see that stop words are gone, you know contractions are gone, punctuation special characters are gone. So, you can see this is quite nicely this has been you know transformed. Now you can compare this with the you know regional version. So, you can see here and you can compare this one and the previous output.

And you can see how the text has been processed, you can see brackets, slash and all those stop words I and another things and commas and all those things are gone. So, you can see how well this has been transformed, same code that we have been discussing in previous few lectures has been used in a using by creating functions and we have been able to you know transform our corpus. Now let us move to the next aspect that is feature extraction, so as we talked about essentially we would like to arrive at a tabular format where colons would be representing terms and vectors and the you know rows could be representing documents.

(Refer Slide Time: 07:51)



jupyter Session Python NumPy Last Checkpoint: 27/03/2019 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
 420346229,  431119827, 1549981252, 3828596596, 4016100855,
1260369986, 2185661387,  906856130, 2638887619,  329987294,
 271912793, 2727656724,  220137690, 1788926742,  724175710,
1980991063, 1785048888, 1233624537,  376442967,  535251329,
 610386602, 2699262066, 1227241057,  623764229, 1991019541,
2008926754, 2510655337,  328333979,  621697005, 1536446999,
4217424823,   23740149, 2196782666, 3251541262, 1208759418,
```

```
In [389]: ## Example: random walks
          # a simple random walk starting at 0 with steps of 1 and -1 occurring with equal probability

          # using Python built-in capabilities
          import random
          x = 0
          walk = [x]
          nsteps = 1000
          for i in range(nsteps):
              step = 1 if random.randint(0, 1) else -1
              x += step
              walk.append(x)

In [390]: walk

Out[390]: [0,
           1,
           0,
           1,
```

So, for that we need to understand feature extraction process, so you know essentially we are talking about vector space model or turn vector model. So, will be transforming and representing text documents as numeric vectors of a specific terms that form the vector dimension. So, these specific terms are extracted features, so these terms are nothing but the extracted features they would be on the colon side and these text documents would on the rows side.

So, now in this particular section we would be talking about you know how to perform this feature extraction process. So, let us take an examples, so we will take a training data set corpus here, so you can see this is the corpus that we are taking few sentences there in this list. And then test data sets 1 sentence there, so you will be you know looking at using this you know training and test data set to perform our feature extraction process.

So, let me define these you know documents is data sets, now first model that is typically use is bag of words model, this is the simplest vectorization model. So, in this to perform this we will define a function to create a bag of words feature you know extraction or vectorization model. So, for this we would be requiring this S key SK learn (()) (11:35) you know this module.feature_extraction.text and we will importing count vectorizer.

So, will be using an instance of this count vectorizer class from here, so we are defining our bow that is bag of words extractor. So, first argument is corpus the you know correction of documents that would be passing on and n gram range the kind of terms that we want. So, in this n gram we can specify a range from unigram to you know trigram or even more.

So, one is indicates first one indicates the starting you know n gram range the second one indicates the ending n gram range. So, you have both are once essentially we are focusing just on the unigrams. So, first thing within this function we need to define this an instance of this you know count vectorizer class and within the arguments we are specifying mean_df as 1.

That is that means we want terms having minimum frequency of one, so that means you know whatever terms we would like to extract features, we would like to extract, this would have at least this much frequency. Then n gram range the same we are using we are just focusing on unigrams here, then using this instance you know vectorizer will be calling this fit_transform method .

And will pass on the corpus here and it will extract us the features in a term document matrix. So, this matrix is actually in pars matrix that would be obtained later will have to convert we have want have a look at the matrix in a regular form regular matrix form. Then we will have to may you know densify it, so that will see, so let me first define this bow extractor. Then next thing we are calling this bow extractor function here to build our vectorizer and get features.

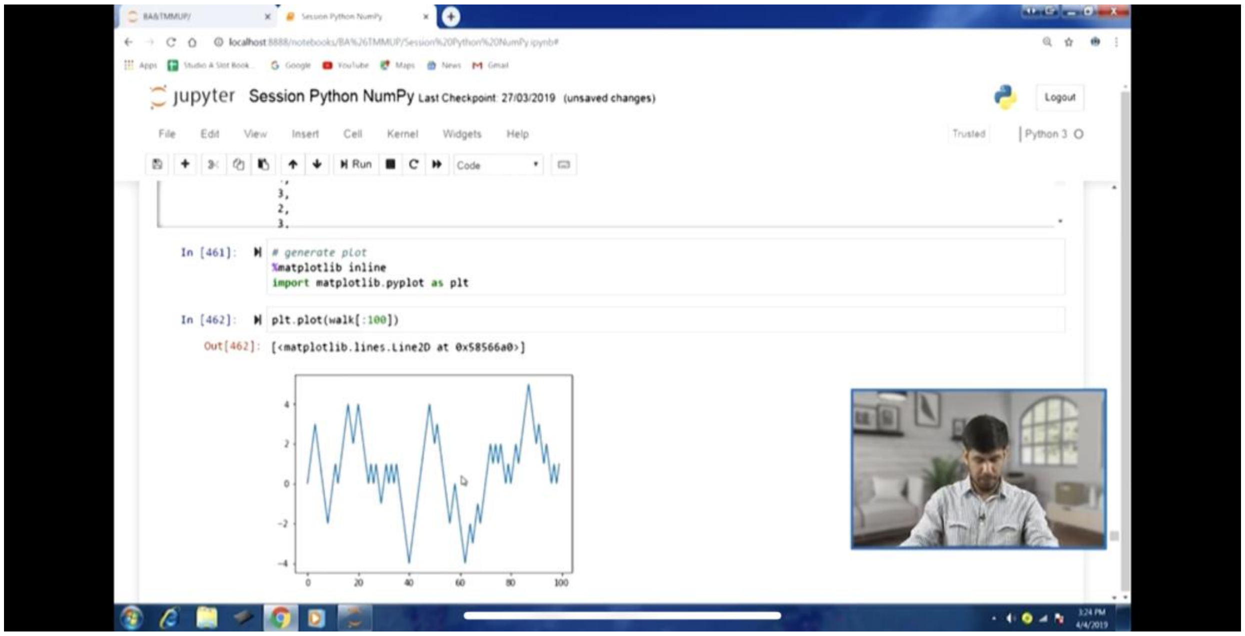
So, we are passing on corpus, so let me run this here. Let us have a look at features we can see this is 4*15 pars matrix here. So, 4 rows 15 columns and similarly we can obtain you know convert into a matrix form, so that you know so for this we are using 2 dense method here, so let us have a look you can see. This is the matrix form the tabular kind of form right now in the matrix later will convert into a data frame.

So, that the understandability of this datas would be much easier. So, you can see 4 you know 4 rows here and 15 about 15 columns , so this is the this is what these are the this how we can extract terms or features you know from our corpus. Now we have already defined our vectorizer instance here, now using that we can also transform the test datas are there that is unseen doc that we are created.

So, using that we can also transform this one again you can create a matrix regular matrix here also. So, you can see because we had just a 1 sentence there, so just row here. Now we can also have a look at the feature name, so for that we can call this get_feature_names method. So, if I run this and let us have a look at feature names you can see these are the features, these have been extracted from the corpus text.

So, these are these feature for going to columns and the 4 sentences they would be representing the 4 documents, text document on the row side. So, for a better understanding of this data set let us use this data frame, so will combine feature vectors into a data frame now. So, in the data frame function, we are passing feature_cm and feature_name. So, if I run this we will have this data frame you can see.

(Refer Slide Time: 09:37)



jupyter Session Python NumPy Last Checkpoint: 27/03/2019 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
         3,
         2,
         3,
```

```
In [461]:  # generate plot
           %matplotlib inline
           import matplotlib.pyplot as plt
```

```
In [462]:  plt.plot(walk[:100])
Out[462]:  [<matplotlib.lines.Line2D at 0x58566a0>]
```

So you can see on the column side we have all the terms and, are, brown, clouds, dark, due, specters all these terms that were part of the corpus example that we had and 4 rows 0, 1, 2, 3. So, you can see this much how we have been able to convert we have been able to extract features. And you can see the tabular layout how we have been able to convert and the values these are nothing but frequencies.

So, this is based on bag of words, so frequency now let us have a look at the you know test document that we have. So, unseen document, so let me get this you can see just 1 row and same features. Now let us move to the more important model, more important vectorization model that is tfidf model. So, this particular model solves an important you know problem of you know bag of word model.

So, there you know words with higher frequencies you know they might dominate. Because they might have higher frequencies across documents, how about some of the more relevant words. So, more relevant words in the sense bill if we are doing a text classification problem. So, there are going to be certain process that are going to be you know, so documents are going to be categorize into those classes.

So, certain words might be more relevant for this process but they might have lower frequency. So, in bag of words model that problem is there that relevant words might have lower weightage the you know meaningless words with higher frequency might end of having higher weightage. So, to get rid of that problem we have this tfidf you know model, so for this we are defining this function to create tfidf feature extraction or a vectorization model based on bow features.

So, whatever bag of words features we have extracted, so from that we are trying to transform it into a tfidf feature. So, within this we are using tfidf transformer and instance of this there we have to specify in the first argument the normalization that we want to perform. So, l2 means euclidean kind of normalization, smooth idf you know here you know we would like to you know in the document frequency we would like to add 10.

So, that we are able to prevent 0 divisions use idf is true, so this will define our you know this transformer tfidf transformer. Then we will call fit _ transform method on bag of words features. And that will transform these bag of word features into tfidf features, so this is the function. So, this function takes bag of word features and transform them into tfidf features. So, let us define this then you know let us you know use this function to you know create to transform bag of words features into tfidf features.

(Refer Slide Time: 12:44)

jupyter Session Python NumPy Last Checkpoint: 27/03/2019 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [467]:  # minimum value along the walk's trajectory
           walk1.min()
Out[467]:  -7

In [398]:  # maximum value along the walk's trajectory
           walk1.max()
Out[398]:  27

In [399]:  # first crossing time: step at which the random walk reaches a particular value
```

So, this is what we are doing now, so transform_c features_ct you know we are calling tfidf_transform here. We are passing features_c which are actually you know bag of word features we which we had you know earlier created. Then next thing is we are you know creating the regular matrix too dense method and then we are creating the data frame we are combining the you know tfidf feature vectors into data frame. So, in 1 go will be doing all this and you will see that the values will change.

Now instead of earlier we had zeros and ones, now we have you know these you know because this has been normalized because we are applying tfidf model, so the values have changed. So, this is more useful that is you know typically used in text classification, formation, extraction and many other you know applications. Now we can use this transform method to extract from unseen documents.

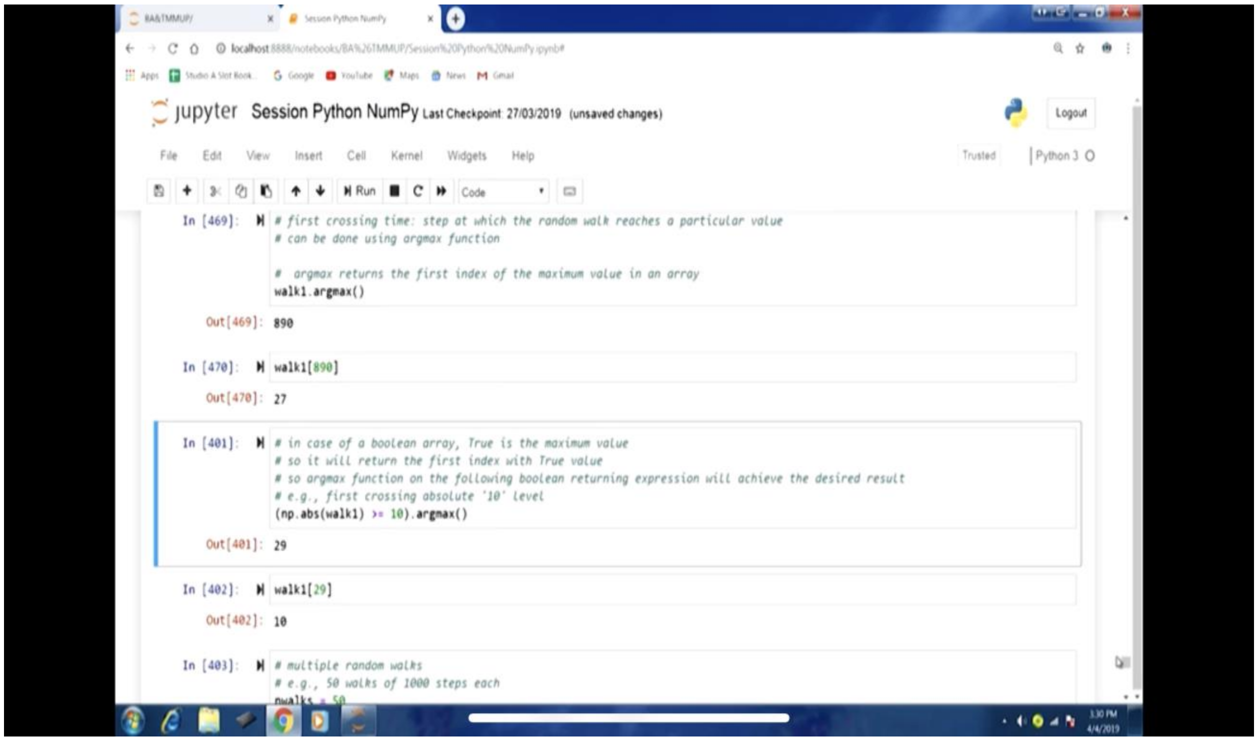
So, again here we are using this transformer_c class and calling transform function to the features for unseen document. Then densifying it for the matrix form and then we are combining it into a data frame. So, if I run this you can see we have the data frame for the test data set that we had. Now let us move forward, so the previous you know transformer tfidf transform that we talked about.

That was based on the bag of word feature, what if want we can also create you know these tfidf features directly from the or perform tfidf vectorization directly from the raw documents. So, for that tfidf vectorizer class and instance of that, so now we are going to talk about another function def tfidf_extractor. First document is corpus, next document is to specify the n gram range just like we did in the bow model.

So, here again will specify will define our class distances tfidf vectorizer class instance, minimum frequency, mean def norm, normalization procedures, smooth idf just to as we discussed to prevent zeros, divisions and use idf and range. And then will be calling fit_transform method to escalate stacked feature based on the vectorizer that we just defined.

So, let me define this and then now let us apply this tfidf extractor on the corpus and will have our vectorizer and features tfidf features. Now again let us convert you know these features you know this is pars matrix into a dense matrix and then into a data frame. So, you can see this is the output, so this is directly by using the extractor function where we are directly extracting the features from the raw documents.

(Refer Slide Time: 16:50)


jupyter Session Python NumPy Last Checkpoint: 27/03/2019 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [469]:  # first crossing time: step at which the random walk reaches a particular value
           # can be done using argmax function

           #  argmax returns the first index of the maximum value in an array
           walk1.argmax()

Out[469]:  890

In [470]:  walk1[890]

Out[470]:  27

In [401]:  # in case of a boolean array, True is the maximum value
           # so it will return the first index with True value
           # so argmax function on the following boolean returning expression will achieve the desired result
           # e.g., first crossing absolute '10' level
           (np.abs(walk1) >= 10).argmax()

Out[401]:  29

In [402]:  walk1[29]

Out[402]:  10

In [403]:  # multiple random walks
           # e.g., 50 walks of 1000 steps each
           nwalks = 50
```

So, you can see the output if we compare this output with the previous one, you will see the same numbers are there. The only difference is the earlier one was from the you can see here the earlier one was from the bag of word feature this one is computed from the raw documents. Now we can apply this on the unsing documents also, so let me run this and we can see in the output, same output and this is directly from the raw documents.

Now let us move forward, now with this we have been able to reach, so all till now what we have been able to cover. We have been able to transform the unstructured text into a structured form. So, all these steps that we talked about whether tokenization, lemmatization and you know stop word removals and many other things and then extracting features. So, essentially all those steps you know word to actually transform text into a structured form.

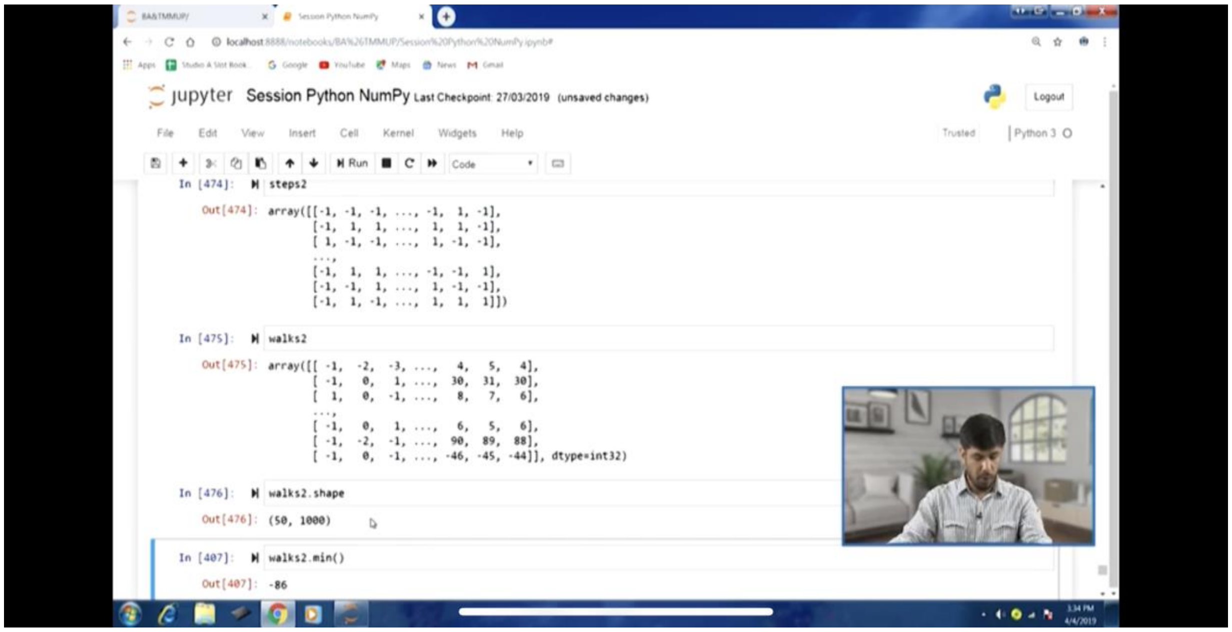
And you could see the data frame, you could see the column, terms the relevant words you know that you know we can extract and in a you know you know in a matrix kind of layout in a tabular layout. So, we have been able to reach that place till now, now what will do will take an example a text classification problem to see how we can build you know text classification models now.

So, for this will be using this 20 news group data set, so this data set it has around 18,000 news group post spread across 20 different categories or topics. So, essentially this is 20 class classification problem, so will be classifying these 18,000 news group post into 1 of these 20 categories. So, for this we first we need to import this data set**,** so for this we need again sklearn.data sets and we will import fetch_20 news groups, this is the data set.

So, for this we are defining this function get_data, so will be using this fetch_20 news groups and certain arguments you would like to remove the headers, footers and codes from the text. So, that only the relevant text is there which will be processing later, so let me define this get_data function. Now we will use this one to obtain the data you can see the download process as starter.

So, once it completes then will be using this data set to build our classifier, so these post are to be classified into one of the 20 categories that are there. And we will see how all the you know transformation steps that we have discussed how they are going to be used here. So, it is still running, you can see here in the in and within brackets asterisk is there, so still asterisk is there till a number is assign there.

(Refer Slide Time: 19:18)


jupyter Session Python NumPy Last Checkpoint: 27/03/2019 (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Logout
Trusted
Python 3

```
In [474]:  steps2

Out[474]: array([[-1, -1, -1, ..., -1,  1, -1],
                 [-1,  1,  1, ...,  1,  1, -1],
                 [ 1, -1, -1, ...,  1, -1, -1],
                 ...,
                 [-1,  1,  1, ..., -1, -1,  1],
                 [-1, -1,  1, ...,  1, -1, -1],
                 [-1,  1, -1, ...,  1,  1,  1]])

In [475]:  walks2

Out[475]: array([[ -1,  -2,  -3, ...,   4,   5,   4],
                 [ -1,   0,   1, ...,  30,  31,  30],
                 [  1,   0,  -1, ...,   8,   7,   6],
                 ...,
                 [ -1,   0,   1, ...,   6,   5,   6],
                 [ -1,  -2,  -1, ...,  90,  89,  88],
                 [ -1,   0,  -1, ..., -46, -45, -44]], dtype=int32)

In [476]:  walks2.shape

Out[476]: (50, 1000)

In [407]:  walks2.min()

Out[407]: -86
```

That means that particular you know code is still running, so let it download. Next thing is you can have a look at the type of data set also once in the download is completes and then will have a look at the classes class names that are there, target name. So, our target variables what are the 20 classes that are there, what are the names there, so we will have a look at that also, this data is still being downloaded.

So, you will see you know many of you know the task that will be performing now they are going to be performed on full corpuses. So, they will take certain times, the idea is to demonstrate you whatever code we have till now discuss how that is you know going to work you know a real problem where we are using a full corpus and building a you know full grown model.

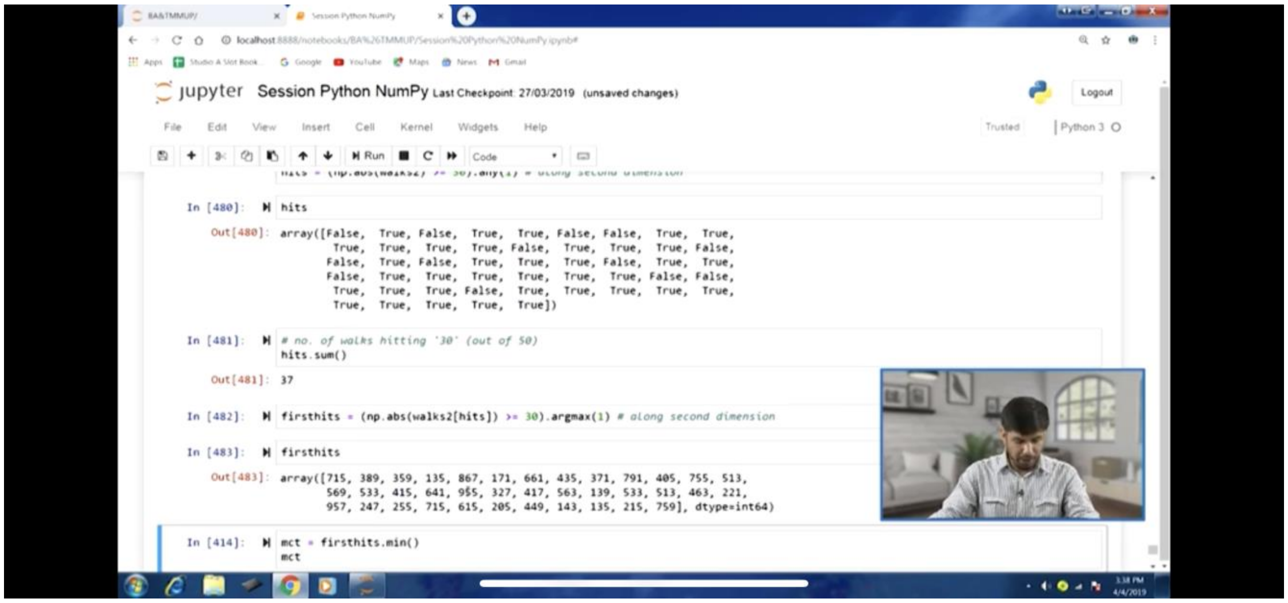
So, **so** downloading has complete, you can see 135 is assign, now let us have a look at the data set you can see sklearn utils.bunch. Now we will print the target names, so essentially these are the classes, so you can see here alt at this con.graphics. So, sci.med, sci.space, sci.crypt, so these are different target names, these are different categories, different classes where these you know documents, these news group post are to be classified.

So, let us move forward, now obtain the corpus and labels, so from this data set that we have just you know obtain will you know create our corpus of documents and labels as well, labels for the target variable, so let me run this. Now the next thing that will be doing is that at a particular corpus might have a number of empty text documents . So, you know mean times you would like to remove them.

So, for this we are defining this function remove_empty_docs, so first argument is corpus, second argument is labels. So, again within this we are defining our corpus_f as a empty list, labels_f as an empty list. Now we are running this for loop doc you know label lbl in zip corpus comma labels and we are using this strip method to find out whether document is you know empty or not.

If it is not empty will append it labels same case so in this fashion will be creating our filtered corpus in filtered labels. And will be removing empty text documents, let me define this function we can apply this here on our corpus. So, let me run this and let us have a look at the corpus here you can see this is the corpus that we have after the removal process.

(Refer Slide Time: 23:10)



```
In [480]:  hits

Out[480]:  array([False,  True, False,  True,  True, False, False,  True,  True,
                   True,  True,  True,  True, False,  True,  True,  True, False,
                  False,  True, False,  True,  True,  True, False,  True,  True,
                  False,  True,  True,  True,  True,  True,  True, False, False,
                   True,  True,  True, False,  True,  True,  True,  True,  True,
                   True,  True,  True,  True,  True])

In [481]:  # no. of walks hitting '30' (out of 50)
           hits.sum()

Out[481]:  37

In [482]:  firsthits = (np.abs(walks2[hits]) >= 30).argmax(1) # along second dimension

In [483]:  firsthits

Out[483]:  array([715, 389, 359, 135, 867, 171, 661, 435, 371, 791, 405, 755, 513,
                  569, 533, 415, 641, 955, 327, 417, 563, 139, 533, 513, 463, 221,
                  957, 247, 255, 715, 615, 205, 449, 143, 135, 215, 759], dtype=int64)

In [414]:  mct = firsthits.min()
           mct
```

Let us have a look at the labels you can see the labels, so labels you know each number is indicating the you know of the class name that we just you know saw in the previous output. So, 10 class you know that is you know 10th class, 3rd class and you know in that sense. So, it is 0 index again and here you can have a look at the you know documents.

So, you can see this is a list of you know in a sense this is a list of a string values you can take in that form, you can see comma here and next you know document is starting. So, in this fashion they have been labeled also, so we got the corpus of text documents and you know labels also. Now let us have a look at the let us take an example, let us take example document and will also have a look at label index and class.

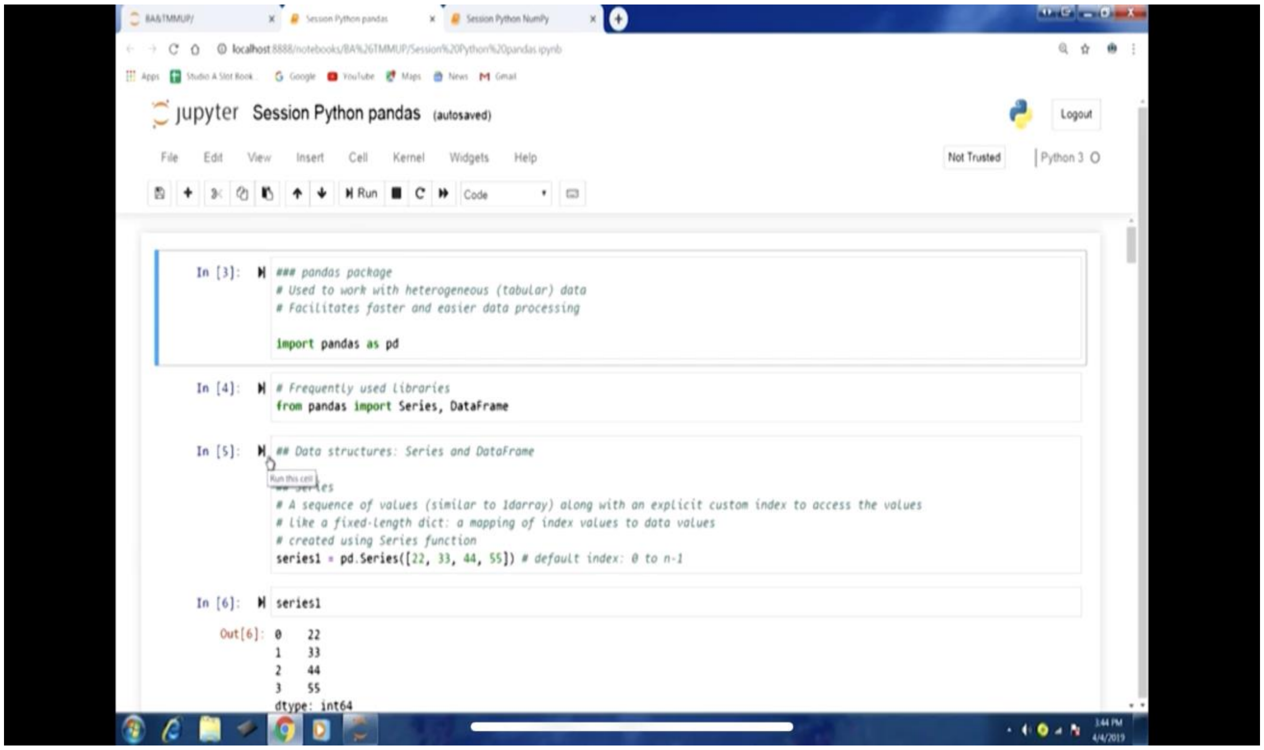
So, document with index value 10, so let us have a look at here corpus 3, so this is the you know fourth text document that is there you can check it back there and let us have a look at it is label 3 and you can see the name. So, if I go back you can see it is ibm.pc.hardware, so if I go back to the list that we had earlier generator, so you can see it will be at fourth place you can see here, you can see com.sys.ibm.pc.hardware.

So, you can see the corpus document and their labeling we have been able to obtain and 1 sample document we have been able to crosscheck in this fashion. Now let us move forward, now next thing as we discuss in our previous courses business analytics and data mining modeling using r typically partition of a data set into training and test. So, here will be using sklearn.model and course selection module and import this train_test_split function to perform this partitioning process.

So, again we are writing our own wrapper function here, prepare_data sets, corpus first argument labels, second argument and the proportion is the 3rd argument that proportion. Then within this we are calling this strain as_test_split function and specifying you know certain arguments here and this will give us the partitions. So, let me define this, so let us call this you know function prep_data sets and we will have our training_c that is corpus training data part, test part, test_t test partition.

Then same for labels also, so this is done, so you can have a look at the train_c this is our training partition. Now test_c this is our test partition, so you can see they have been randomly this you know text documents have been allocated to these partitions. Now we are going to call our transform_corpus function that we had already define. So, in this will be transforming this unstructured text as you can see in those 2 partition training and test partition.

(Refer Slide Time: 26:44)



jupyter Session Python pandas (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code
Run this cell

```
In [3]:  ### pandas package
         # Used to work with heterogeneous (tabular) data
         # Facilitates faster and easier data processing

         import pandas as pd

In [4]:  # Frequently used libraries
         from pandas import Series, DataFrame

In [5]:  ## Data structures: Series and DataFrame

         ## Series
         # A sequence of values (similar to 1darray) along with an explicit custom index to access the values
         # like a fixed-length dict: a mapping of index values to data values
         # created using Series function
         series1 = pd.Series([22, 33, 44, 55]) # default index: 0 to n-1

In [6]:  series1

Out[6]:  0    22
         1    33
         2    44
         3    55
         dtype: int64
```

In to a form that can be later on used in our when we apply our machine learning algorithms. So, transform_corpus for both training_c and test_c both these you know partitions and will be performing expanding contraction POS tagging, lemmatization, special character removals, stop word removal, tokenization. So, all these things are going to be perform this.

So, if I run this it will take lot more time, so will have to pleasantly wait here, so this process because lot of lot more processing will go on. Because this is full corpus that we have taken, so it will take lot more time to actually you know take each document one by one you know and process it, so it will take lot more time here. So, you will have to be wait, so you can see that asterisk is there and that particular you know code that we are executing and it will take time.

So, we will have to wait, it will take minutes and once this transformation process complete then we will move to the next part that is extracting features. And once we are ready with our you know vectorization process over once we are ready with our you know tabular format then we can apply our machine learning algorithms for the classification problem. So, to demonstrate a particular text classification model will be using support vector machines as an example.

So, we would not be going into details of support vector machine that you can refer any you know books related to text mining, data mining and understand that particular method. Here we are just trying to demonstrate how in using python platform and specifically nltk module how we are able to model this. So our focus is on the modeling part and how that can be perform in this you know python platform.

So, once this transformation is over will be executing some more you know steps here. So, you can see this the I have been able to process this, so you can this took I think about 2 to 3 minutes or even more to actually transform these you know these partitions. So, this gives you about the this gives you the idea about how much time it might take in you know processing you know unstructured you know text there.

So, that is how you will also get the idea about big data analytics and how this unstructured data that is there and why it requires big IT investments, big setups to actually process those large amounts of data. So, this is small example itself is demonstrating this aspect, so let us have a look at the train_ct this transform text here, they can see now the nicely transform text is provide visible here, same for the test partition here.

So, now the next thing will be extracting features form training partition. So, first thing bag of words feature, the another model that will use this tfidf feature. So, again just like you know previously done will you know generate bill our vectorizer and obtain features. So, first bow_extractor then will get features for the test data set.

Similarly for tfidf we will use tfidf extractor that we have defined earlier and we will obtain features for the training partition, then will obtain features for the test partition. So, let me run this and will get the features from here, once this is done we will move to will tokenize the documents. So, in this we are running this list comprehension where you can see for each text in the training partition and each text in the text you know test partition also in the next line of code we are tokenizing.

So, let me run this one also and we will have the you know tokenized documents. Now next aspect is evaluation matrix, so whenever we are you know building a classified classification model will also like to you know compute certain matrix to check the performance how well our model has done in terms of you know classification or prediction for that matter as well.

So, we will talk about you know these matrix also, so the matrix are quite similar to what we have discussed in our previous courses business analytics and data mining modeling using r. So, here we are just defining a function get_matrix where we are passing on actual labels and predicted labels and computing accuracy precision recall and fn is code. So, for this we are using this matrix you know module and there we have accuracy_score precision_score, recall_score and fn_ you know function.

And we are using that will be using that to actually you know compute these matrix. So, let me define this function first because later on as we will the classifier will be calling this function. Now next we are defining a function to actually you know build the model using training data set and then evaluate the model performance on test data set. So, in this refer function this is generic kind of function where passing on the classifier.

So this could be based on any technique sbm or neo base or any other thing, any other machine algorithm and we are passing on train_training features you know training labels, test features, test labels. And first you know code is first line of code is about building the model the classifier.fit, passing training features and training labels. Then predicting predict using models, so we are predicting the test partition using the classifier.predict you know function here.

And then will use our user define get_matrix_matrix function and will be evaluating model you know prediction performance. So, let me define this function as well, now as an example we are going to build a classifier using support vector machine here. So, for this we need this SGDclassifier you know class and will create an instance of this class sgj classifier and will define this instance. So, first perimeter loss specified as hinge because we are just building a linear svm, maximum iteration 1000 tol.2.

That is just to define the you know allowed loss there, so let me define this svm . Now we will call our train f_predict_evaluate model to actually build the model as well as actually you know evaluate the performance. So, for you know training you know for that you know this one for using bag of words, so let me run this. So, first one is using bag of word feature you can see accuracy 0.65 and other matrix also.

Now next one is using you know support vector machine with tfidf you know features. So, let me run this and you can see the accuracies 0.77, so accuracy we compare with the bag of words this has gone up, so that is why tfidf is more common more popular vectorization model when we require to build you know when we whenever we require to extract features. Now we can also create generate our classification matrix, remember this was 20 class you know classifier model.

So, we will use our pandas you know function here matrix.confusion_matrix will pass on test_l we saw the actual labels for the test partition. And our prediction tfidf predictions here, so if I run this then we call data frame here to get a nicer output here. And you can see we have a 20/20 matrix and in our previous course business analytics data mining modeling is using are typically use to take 2 classes and this was 2 class 2 matrix. Typically here it is 20/20 matrix here and you can see.

So in terms of understanding this matrix you can see you know class 0 correctly classified as 0136 times and twice classifier as 1, twice classified as you know class you know class with index 2. And similarly for that class 1 correctly classified as class 1 to under 9 times, so diagonal values you see they are on the higher side. So, that is indicating that the class has been correctly classified into it is actual class other numbers are actually misclassification. So, this 20/20 matrix is actually depicting full performance of the model.

So, this with this we have been able to cover our you know text mining modeling part also. This was mainly using nltk module in you know in combination with other modules other functions as well.
(Video ends: 41:45)

So, with this we have been able to you know complete last lecture in this course as well and you know in this course we have been able to cover the introductory part of text mining what can problems are there and what kind of you know concepts are there. So, briefly we were able those aspects in the initial few lectures then we started our discussion on python. Because the knowledge of python extensive knowledge of python is actually required for the analytics especially for the text mining modeling.

So, lot more number of lectures were actually dedicated for you know the python there. Then once a major junk of python was covered which is really went for the analytics. Then we came to you know our text mining modeling and there again in text mining modeling a lot more time is devoted and processing unstructured text. So, you know lot more steps for processing text we were discussed as we have been you know in last few lectures they were dedicated for this.

And in this last lecture using a particular you know we have been able to build a svm model, a classifier model as well. So, that completes our journey in the sense we have been able to you know talk about text mining modeling and python for analytics and also you know the text mining text processing part the transformation part.

And also in this particular lecture we were able to demonstrate a particular one example, 1 text mining modeling example where we build this classifier for the 20 class scenario. So, with this we complete you know our course and if depending on the response for this course. If there is demand will continue you know discussion on few more aspects of text mining modeling maybe you know a number of other techniques.

And maybe we might also focus on how do you know scrap data from you know using various keys and other aspects like this. However you know for right now thank you for the thank you for being with this course and good luck for the future, thank you.

Keywords: Text mining and modelling, tokenize, lemmatizer function, transformation, data scrapping.