DAAN862: Analytics Programming in Python

Lesson 13: Classification of Text

Lesson 13: Classification of Text

At the end of this lesson, students will be able to:

- Define and apply text classification.
- Perform text classification with Naïve Bayes classifier in scikit-learn.
- Perform text classification with Support Vector Machine classifier in scikit-learn.

By the end of this lesson, please complete all readings and assignments found in the Lesson 13 Course Schedule.

Lesson 13.1: Introduction to Text Classification

What is text classification?

Text data is typically referring as a phrase, sentence or paragraphs of text which can be found from blogs, tweets or anywhere from the web. Text classification, or text categorization, is to classify the text data. In this section, you will be shown how to use supervised approach for text classification. For example, you can classify the rating based on customer reviews on online shopping, analyze the sentiment on financial news, etc..

Text Classification Procedures:

Now you know the basic concept of text classification. Assuming we have our dataset already downloaded and ready to be used, the standard procedures to complete this task are:

- Prepare traing and test datasets
- Text normalization
- Feature extracation
- Model training
- Model prediction and evaluation
- Model deployment

Transcript

In Lesson 13, I will give two text classification example. In this video, I will use a Naïve Bayes model for classifying the sentence type.

First, Let’s import the necessary packages: nltk, pandas, os and numpy.

First, we set the working directory to where you saved sentenceCorpus, and we focus on the labeled articles.

We get all file names by using os dot listdir() function.

I have to pop the first file since I am using google drive.

You can take a look at the first 5 file names.

We initiate two functions we will use: CountVectorize and PorterStemmer

Now we open a file and take a look at the text in each file.

The abstract is the section name. MISC is a sentence type anything else. AIMX is the goal of the paper. OWNX is their own work. You should exam several files to have a basic understanding of the file.

Next is the text mining part. We need to convert those text file into a Dataframe data whose columns will be words appeared in the text file and values will be their count.

We use two for loop for it.

In the outer for loop, we go through all files in the folder.

We open the file and read the text into python and call it text.

Then I remove some useless words.

The next step to split the text into sentences.

For each sentence in the text, I will check if the first word belongs to the five categories. If yes, I will get the type of the sense is the first word.

Then normalize the words with porter.stem.

Use count_vector to transform the words. Unique_words can be found from get_feature_namess() method. The data is stored in a matrix format, we need to use sum function to get the count. And reshape the data to (1, n). Create a temporary data frame for it.

Join the temp dataframe with output dataframe df. I like to add a print function in the trying process such that I know the progress and if there any problem and I know which one creates the problem.

Now you have a raw dataframe. There are NAs in the data since during the joining, words which cannot be found will be considered as NAs.

We can fill NAs with 0s which means those words don't appear in the sentence.

We can check the data shape, there are 1211 sentences and 2568 words.

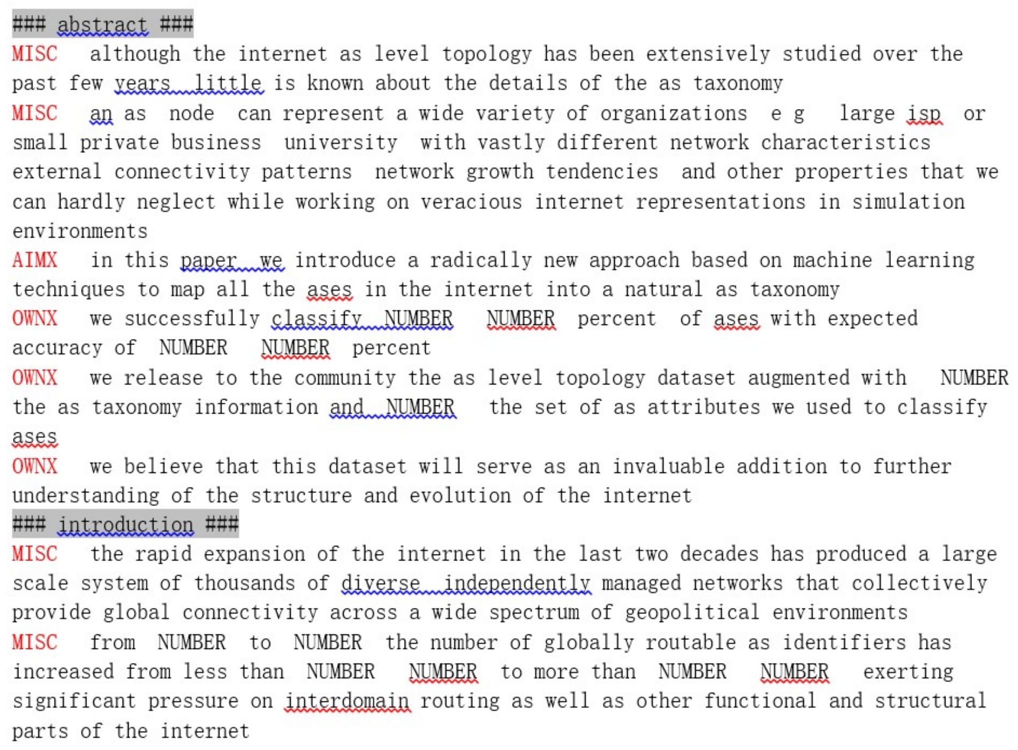
and class value counts.

The rest process is standard model building and evaluation. The accuracy for train and test dataset is 0.90 and 0.77. And these are their confusion matrix.

Lesson 13.2: Text Classification Example 1- Naive Bayes Classifer

Here we show a text classification example use Sentence Classification Data Set (https://archive.ics.uci.edu/ml/datasets/Sentence+Classification) . After you download the dataset folder from UCI website and unzip it, the information about the dataset can be found in the readme file. We are going to use all files in the labeled_articles folder, which are 90 in total. Let's take a look at how the text file looks like. Figure 13.1 shows how it looks like when you open the "arxiv_annotate1_13_1.txt" in the Microsoft Word document. The first word in each sentence represents their categories, as indicated in Red. The abstract and instruction used to indicate the sections:

Figure 13.1 shows what the text file looks like



```
### abstract ###
MISC   although the internet as level topology has been extensively studied over the
past few years  little is known about the details of the as taxonomy
MISC   an as  node  can represent a wide variety of organizations  e g   large isp  or
small private business  university  with vastly different network characteristics
external connectivity patterns  network growth tendencies  and other properties that we
can hardly neglect while working on veracious internet representations in simulation
environments
AIMX   in this paper  we introduce a radically new approach based on machine learning
techniques to map all the ases in the internet into a natural as taxonomy
OWNX   we successfully classify  NUMBER   NUMBER  percent  of ases with expected
accuracy of  NUMBER   NUMBER  percent
OWNX   we release to the community the as level topology dataset augmented with   NUMBER
the as taxonomy information and  NUMBER   the set of as attributes we used to classify
ases
OWNX   we believe that this dataset will serve as an invaluable addition to further
understanding of the structure and evolution of the internet
### introduction ###
MISC   the rapid expansion of the internet in the last two decades has produced a large
scale system of thousands of diverse  independently managed networks that collectively
provide global connectivity across a wide spectrum of geopolitical environments
MISC   from  NUMBER  to  NUMBER  the number of globally routable as identifiers has
increased from less than  NUMBER   NUMBER  to more than  NUMBER   NUMBER   exerting
significant pressure on interdomain routing as well as other functional and structural
parts of the internet
```

Fig 13.1 (click to enlarge)

There are a total of 5 categories listed in Table 13.1.1 below:

Table 13.1.1: Text Classifications

Aim	AIMX	The specific research goals of the paper.
Own	OWNX	The authors own work, e.g. methods, results, and conclusions.
Contrast	CONT	Contrast, comparison, or critique of past work.
Basis	BASE	Past work that provides the basis for the work in the article.
Misc	MISC	Any other sentences.

You can refer to the manual in Table 13.2 on how to label sentences:

Click dropdown to view manual:

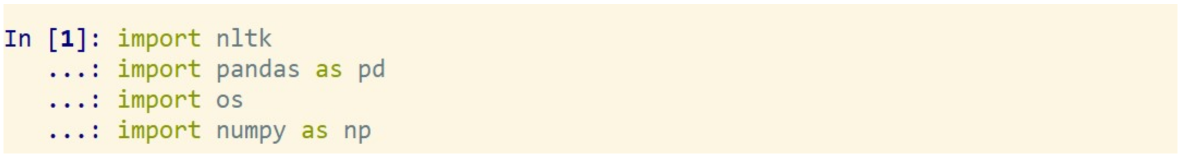
Fig
13.2

For details of how to create these categories from sentences, please referr to the Instructions_for_SentenceAnnotation.pdf file in the folder.

(click the titles to learn more)

Prepare the Data

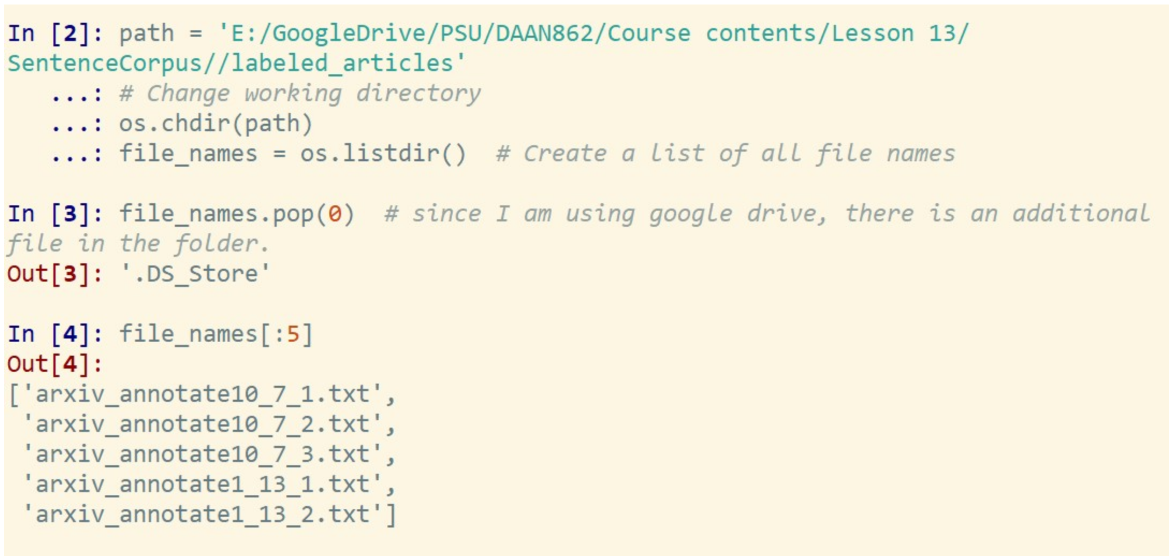
First, ready all filenames in Figure 13.3:



```
In [1]: import nltk
   ...: import pandas as pd
   ...: import os
   ...: import numpy as np
```

Fig 13.3 (click to enlarge)

We change the working directory to train folder by using os.chdir() (shown in Figure 1.34) and read all file names into Python with os.listdir():

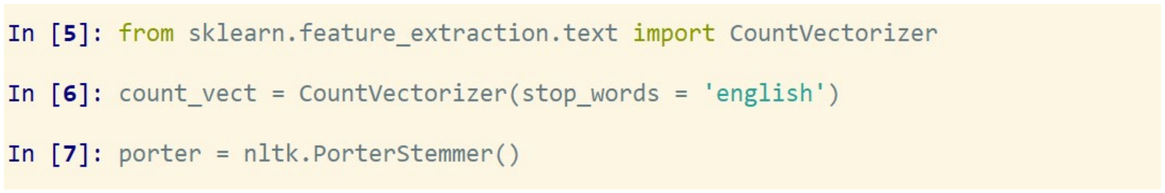


```
In [2]: path = 'E:/GoogleDrive/PSU/DAAN862/Course contents/Lesson 13/
SentenceCorpus//labeled_articles'
   ...: # Change working directory
   ...: os.chdir(path)
   ...: file_names = os.listdir()  # Create a list of all file names

In [3]: file_names.pop(0)  # since I am using google drive, there is an additional
file in the folder.
Out[3]: '.DS_Store'

In [4]: file_names[:5]
Out[4]:
['arxiv_annotate10_7_1.txt',
 'arxiv_annotate10_7_2.txt',
 'arxiv_annotate10_7_3.txt',
 'arxiv_annotate1_13_1.txt',
 'arxiv_annotate1_13_2.txt']
```

Fig 13.4 (click to enlarge)

Text Normalization and Feature Extraction

Next, we use for loop to read all text in from the files, Figure 13.5 shows how we can use a data frame to store words and its frequency:

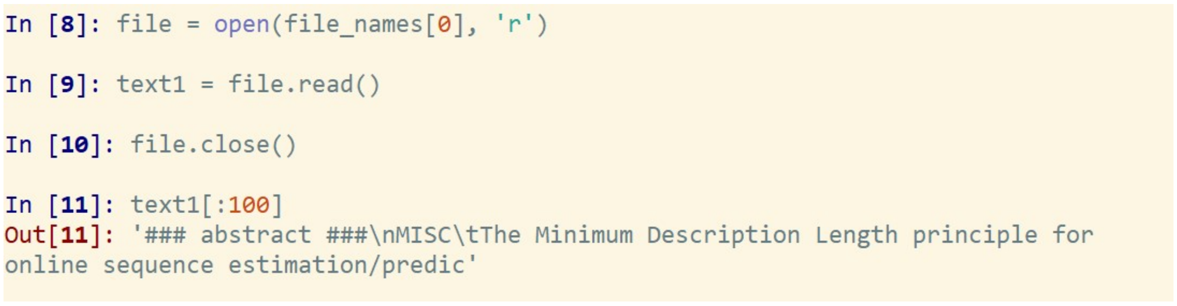


```
In [5]: from sklearn.feature_extraction.text import CountVectorizer

In [6]: count_vect = CountVectorizer(stop_words = 'english')

In [7]: porter = nltk.PorterStemmer()
```

Fig 13.5 (click to enlarge)

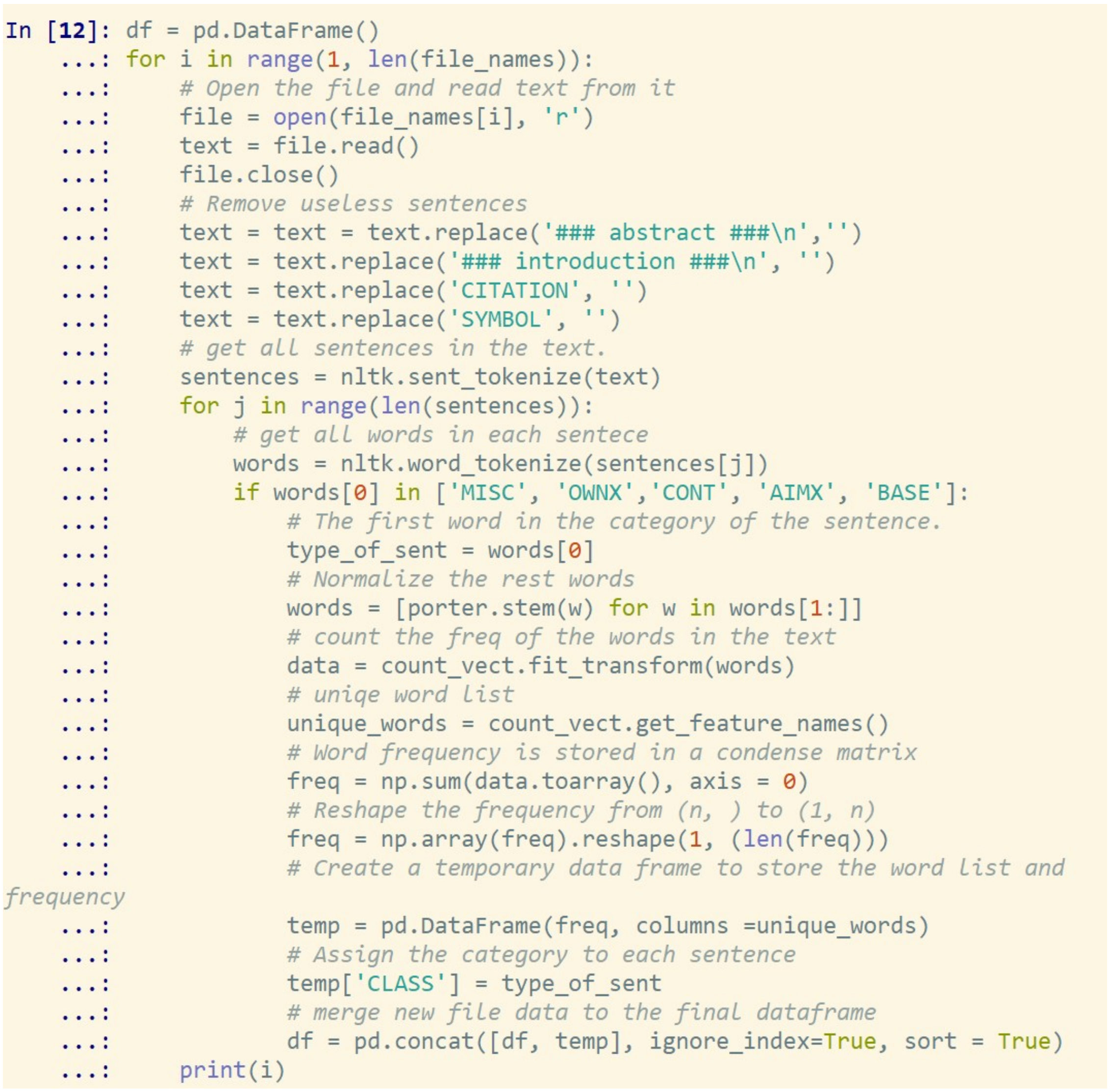
CountVectorizer is used to count the frequency of words as shown in Figure 13.6 (A&B):

Click dropdown to view examples:



```
In [8]: file = open(file_names[0], 'r')

In [9]: text1 = file.read()

In [10]: file.close()

In [11]: text1[:100]
Out[11]: '### abstract ###\nMISC\tThe Minimum Description Length principle for
online sequence estimation/predic'
```

Fig 13.6 (A) (click to enlarge)



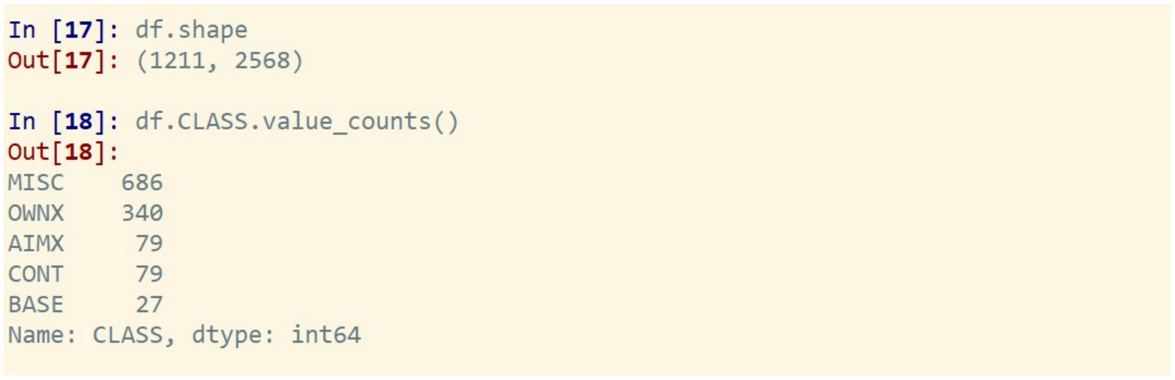
```
In [12]: df = pd.DataFrame()
    ...: for i in range(1, len(file_names)):
    ...:     # Open the file and read text from it
    ...:     file = open(file_names[i], 'r')
    ...:     text = file.read()
    ...:     file.close()
    ...:     # Remove useless sentences
    ...:     text = text = text.replace('### abstract ###\n','')
    ...:     text = text.replace('### introduction ###\n', '')
    ...:     text = text.replace('CITATION', '')
    ...:     text = text.replace('SYMBOL', '')
    ...:     # get all sentences in the text.
    ...:     sentences = nltk.sent_tokenize(text)
    ...:     for j in range(len(sentences)):
    ...:         # get all words in each sentece
    ...:         words = nltk.word_tokenize(sentences[j])
    ...:         if words[0] in ['MISC', 'OWNX','CONT', 'AIMX', 'BASE']:
    ...:             # The first word in the category of the sentence.
    ...:             type_of_sent = words[0]
    ...:             # Normalize the rest words
    ...:             words = [porter.stem(w) for w in words[1:]]
    ...:             # count the freq of the words in the text
    ...:             data = count_vect.fit_transform(words)
    ...:             # uniqe word list
    ...:             unique_words = count_vect.get_feature_names()
    ...:             # Word frequency is stored in a condense matrix
    ...:             freq = np.sum(data.toarray(), axis = 0)
    ...:             # Reshape the frequency from (n, ) to (1, n)
    ...:             freq = np.array(freq).reshape(1, (len(freq)))
    ...:             # Create a temporary data frame to store the word list and
frequency
    ...:             temp = pd.DataFrame(freq, columns =unique_words)
    ...:             # Assign the category to each sentence
    ...:             temp['CLASS'] = type_of_sent
    ...:             # merge new file data to the final dataframe
    ...:             df = pd.concat([df, temp], ignore_index=True, sort = True)
    ...:     print(i)
```

Fig 13.6 (B) (click to enlarge)

Now, that NA's have been introduced in concat functions; let's try to fill it with 0 as shown in Figure 13.7:

Fig 13.7 (click to enlarge)

Now, we will perform data cleaning. Replace NA with 0 for the joined dataframe (shown in Figure 13.8):

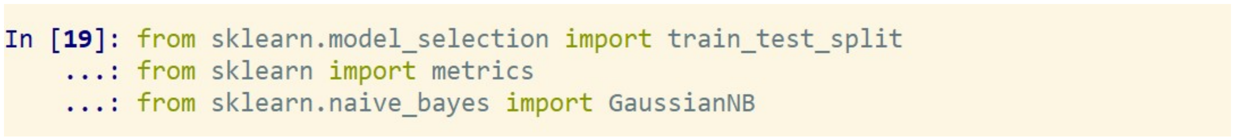


```
In [17]: df.shape
Out[17]: (1211, 2568)

In [18]: df.CLASS.value_counts()
Out[18]:
MISC    686
OWNX    340
AIMX     79
CONT     79
BASE     27
Name: CLASS, dtype: int64
```

Fig 13.8 (click to enlarge)

After performing data cleaning, you now have a clean dataset which uses column names and word frequency as values. The model generation and evaluation portion is exactly the same as the one lesson 8. In Figure 13.9, we use Naive Bayes model for this task, since it's one of the main methods for text classification:

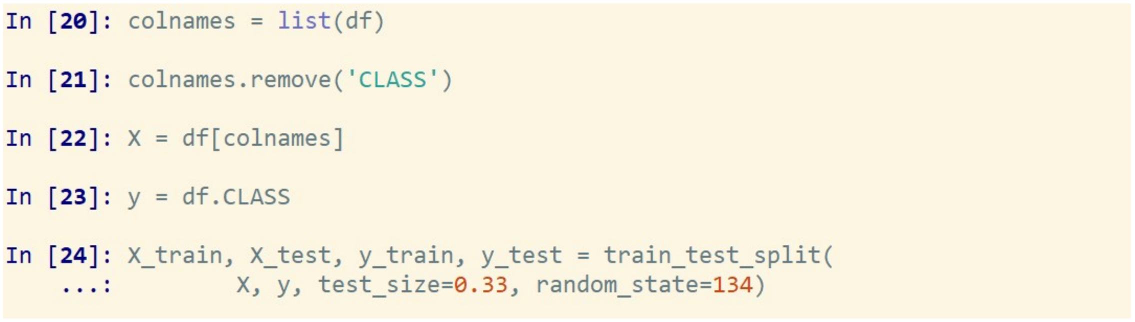
Import objects we need for model generation and evaluation:



```
In [19]: from sklearn.model_selection import train_test_split
    ...: from sklearn import metrics
    ...: from sklearn.naive_bayes import GaussianNB
```

Fig 13.9 (click to enlarge)

Lastly, we separated data into dependent variable X and independent variable y, then divided them into train and test data set shown in Figure 13.10:

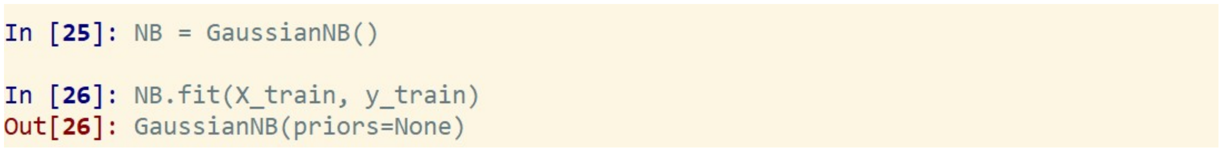


```
In [20]: colnames = list(df)

In [21]: colnames.remove('CLASS')

In [22]: X = df[colnames]

In [23]: y = df.CLASS

In [24]: X_train, X_test, y_train, y_test = train_test_split(
    ...:         X, y, test_size=0.33, random_state=134)
```

Fig 13.10 (click to enlarge)

Build the Model with Naive Bayes Classifier

Figure 13.11 shows an example of Naive Bayes Classifier:



```
In [25]: NB = GaussianNB()

In [26]: NB.fit(X_train, y_train)
Out[26]: GaussianNB(priors=None)
```

Fig 13.11 (click to enlarge)

Model Evaluation

Figure 13.12 (A&B) shows examples of Model Evaluation:

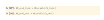
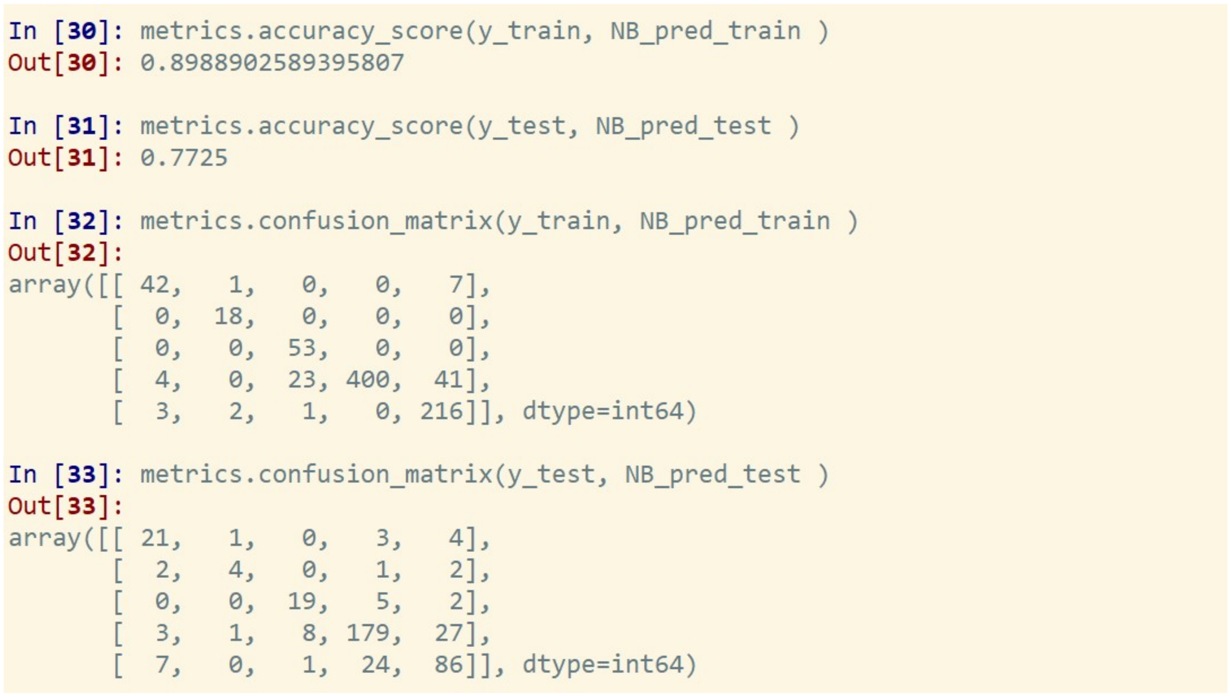


Fig 13.12 (A) (click to enlarge)



```
In [30]: metrics.accuracy_score(y_train, NB_pred_train )
Out[30]: 0.8988902589395807

In [31]: metrics.accuracy_score(y_test, NB_pred_test )
Out[31]: 0.7725

In [32]: metrics.confusion_matrix(y_train, NB_pred_train )
Out[32]:
array([[ 42,   1,   0,   0,   7],
       [  0,  18,   0,   0,   0],
       [  0,   0,  53,   0,   0],
       [  4,   0,  23, 400,  41],
       [  3,   2,   1,   0, 216]], dtype=int64)

In [33]: metrics.confusion_matrix(y_test, NB_pred_test )
Out[33]:
array([[ 21,   1,   0,   3,   4],
       [  2,   4,   0,   1,   2],
       [  0,   0,  19,   5,   2],
       [  3,   1,   8, 179,  27],
       [  7,   0,   1,  24,  86]], dtype=int64)
```

Fig 13.12 (B) (click to enlarge)

Transcript

In this video, I will show hot to use support vector machine for text mining. We will use DBword data, which is a clean dataset. No need to prepare the data on your own.

The data contains 64 instances while 4703 words.

They use 0, and 1 for class labels to determine if the email content is related to announcement of conferences. There are 29 instances are related and 35 are not related.

There are no missing values in the data.

You split the data into X, y then train and test data.

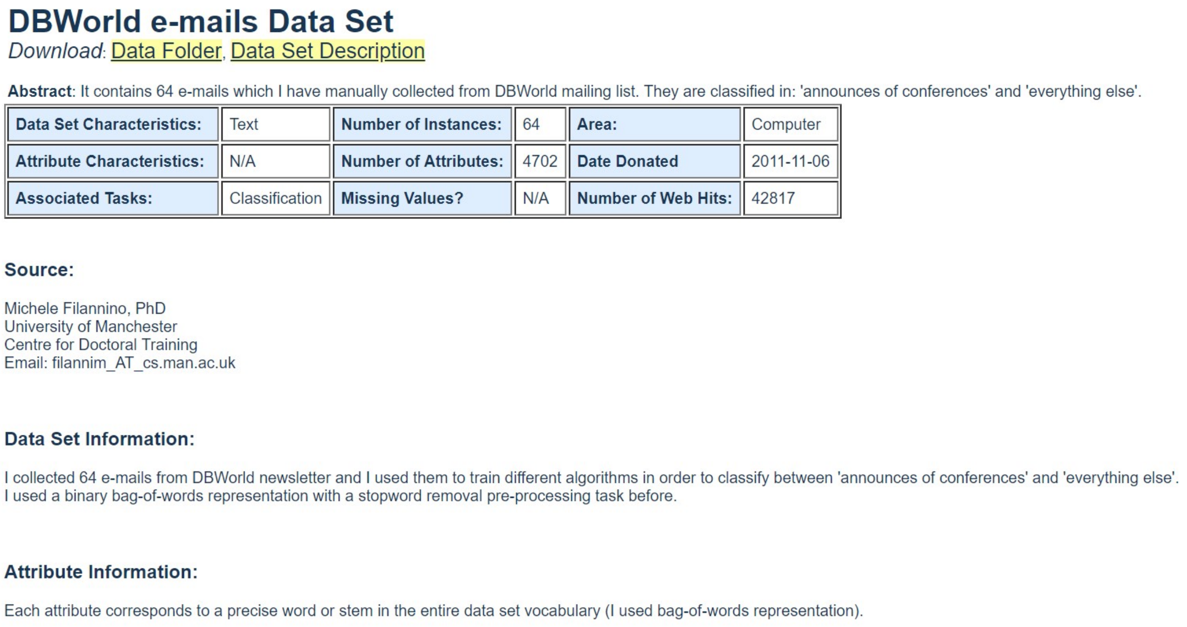
Here we use a linear kernel for support vector machine model.

Next, we fit the model with train data and evaluate both train and test data.

The train accuracy is 1 and test accuracy is 0.91

Lesson 13.3: Text Classification Example 2- Support Vector Machines

For this example, we are going to use *dataDBworld*. Figure 13.13 shows the information about this dataset. You will need to identify if an email is “announces of conferences’ or ‘everything else”.



DBWorld e-mails Data Set

Download: Data Folder, Data Set Description

Abstract: It contains 64 e-mails which I have manually collected from DBWorld mailing list. They are classified in: 'announces of conferences' and 'everything else'.

Data Set Characteristics:	Text	**Number of Instances:**	64	**Area:**	Computer
Attribute Characteristics:	N/A	**Number of Attributes:**	4702	**Date Donated**	2011-11-06
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	42817

Source:

Michele Filannino, PhD
University of Manchester
Centre for Doctoral Training
Email: filannim_AT_cs.man.ac.uk

Data Set Information:

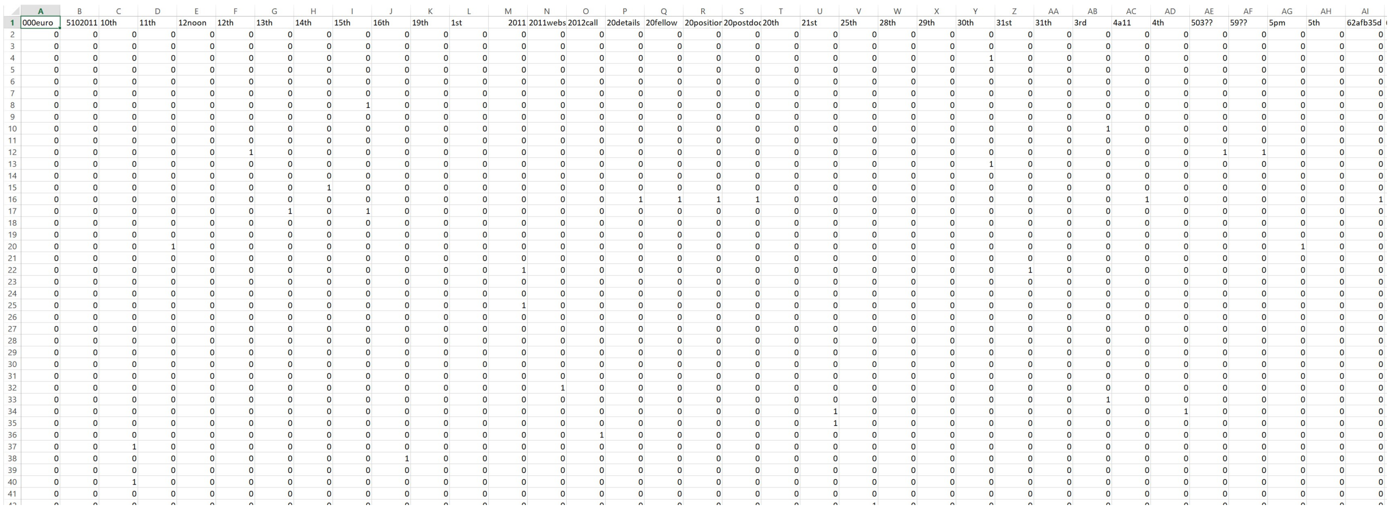
I collected 64 e-mails from DBWorld newsletter and I used them to train different algorithms in order to classify between 'announces of conferences' and 'everything else'.
I used a binary bag-of-words representation with a stopword removal pre-processing task before.

Attribute Information:

Each attribute corresponds to a precise word or stem in the entire data set vocabulary (I used bag-of-words representation).

Fig 13.13 (click to enlarge)

Figure 13.14 shows a a clean version of the dataset:*dbworld_bodies.csv*. Since this is cleaned data, we are going to skip steps 1-3, and instead focus on what the csv file looks like:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI
1	000euro	5102011	10th	11th	12noon	12th	13th	14th	15th	16th	19th	1st	2011	2011webs	2012call	20details	20fellow	20positior	20postdoc	20th	21st	25th	28th	29th	30th	31st	31th	3rd	4a11	4th	503??	59??	5pm	5th	62afb35d
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
17	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

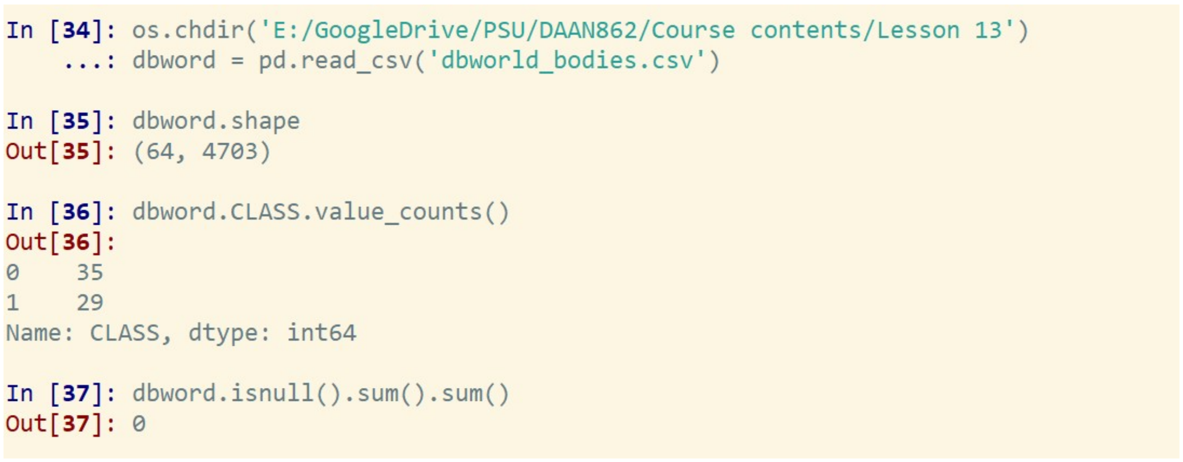
Fig 13.14 (click to enlarge)

Load data into python, Build the model with support vector machine classifier, and Evaluate the model:

(click tabs to learn more)

Load Data Into Python:

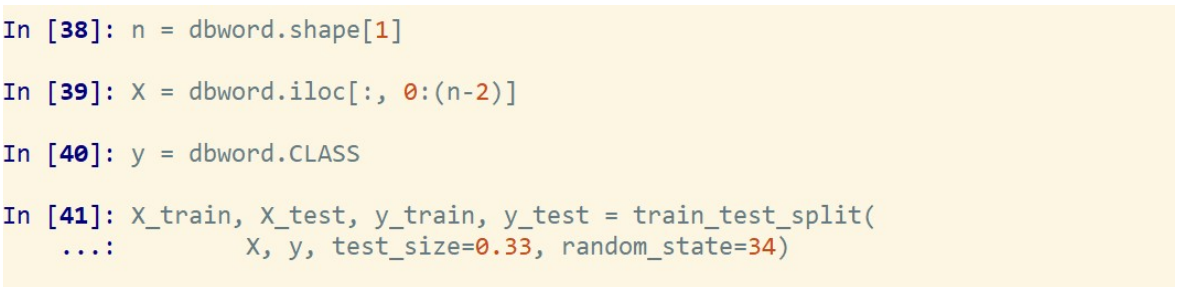
Let’s change the working directory and import this data into Python. In Figure 13.15, we can see that there are 64 emails and 4703 words are used as attributes. There are two categories [0, 1] in Class column, and no missing values:



```
In [34]: os.chdir('E:/GoogleDrive/PSU/DAAN862/Course contents/Lesson 13')
    ...: dbword = pd.read_csv('dbworld_bodies.csv')

In [35]: dbword.shape
Out[35]: (64, 4703)

In [36]: dbword.CLASS.value_counts()
Out[36]:
0    35
1    29
Name: CLASS, dtype: int64

In [37]: dbword.isnull().sum().sum()
Out[37]: 0
```

Fig 13.15 (click to enlarge)

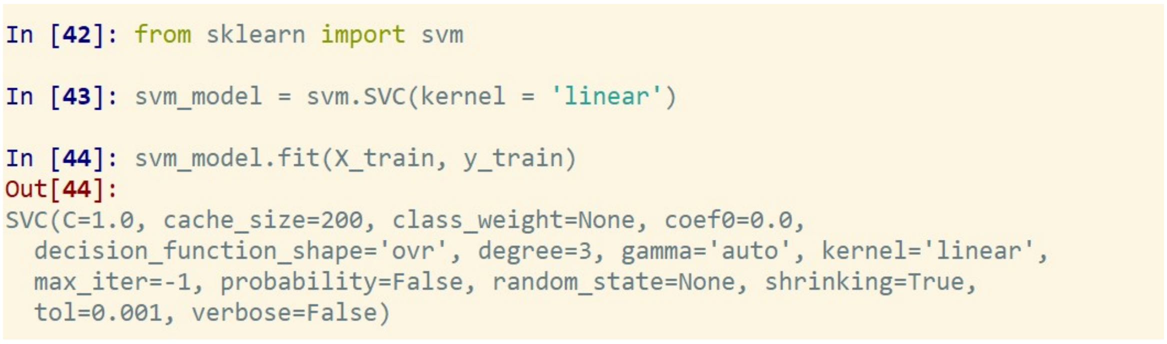
Next, we seperate the data into dependent variable X and independent variable y, then divided them into train and test data set as shown in Figure 13.16:



```
In [38]: n = dbword.shape[1]

In [39]: X = dbword.iloc[:, 0:(n-2)]

In [40]: y = dbword.CLASS

In [41]: X_train, X_test, y_train, y_test = train_test_split(
    ...:         X, y, test_size=0.33, random_state=34)
```

Fig 13.16 (click to enlarge)

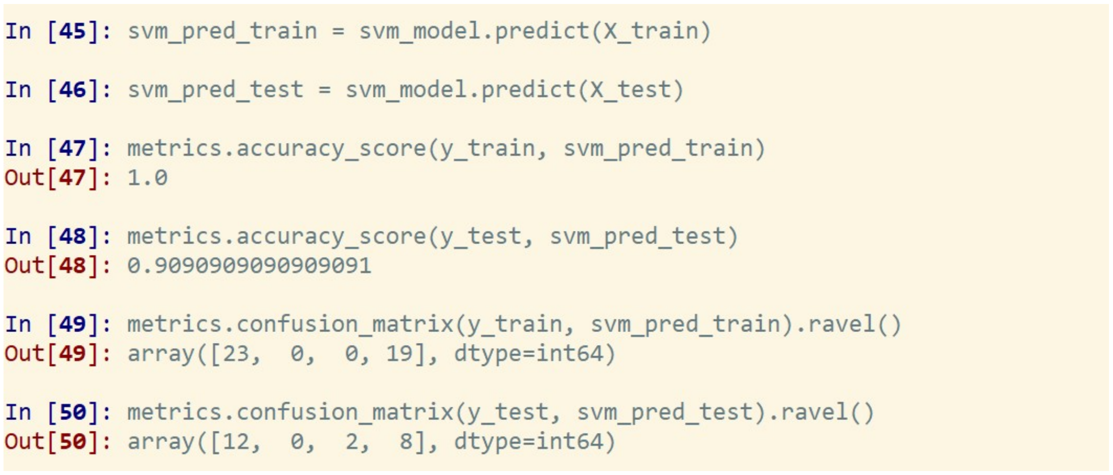
Build The Model Support Vector Machine Classifier:



```
In [42]: from sklearn import svm

In [43]: svm_model = svm.SVC(kernel = 'linear')

In [44]: svm_model.fit(X_train, y_train)
Out[44]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Fig 13.17 (click to enlarge)

Evaluate The Model:



```
In [45]: svm_pred_train = svm_model.predict(X_train)

In [46]: svm_pred_test = svm_model.predict(X_test)

In [47]: metrics.accuracy_score(y_train, svm_pred_train)
Out[47]: 1.0

In [48]: metrics.accuracy_score(y_test, svm_pred_test)
Out[48]: 0.9090909090909091

In [49]: metrics.confusion_matrix(y_train, svm_pred_train).ravel()
Out[49]: array([23,  0,  0, 19], dtype=int64)

In [50]: metrics.confusion_matrix(y_test, svm_pred_test).ravel()
Out[50]: array([12,  0,  2,  8], dtype=int64)
```

Fig 13.18 (click to enlarge)

Lesson 13.4 References

- Chapter 4 “Text Analytics with Python”, Dipanjan Sarkar

Lesson 13 References (5 of 5)

Lesson 13 References

- Chapter 4 “Text Analytics with Python”, Dipanjan Sarkar