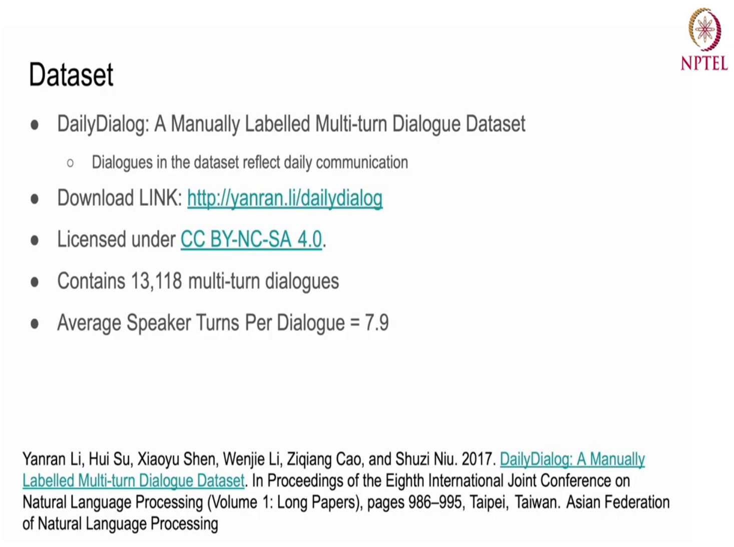
Affective Computing

Dr. Abhinav Dhall
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi

Week - 06
Lecture - 20
Tutorial: on Emotion Recognition using Text

Hi everyone, I welcome you all in this Affective Computing Tutorial on Emotion Recognition using Text. In this tutorial, we will work with a text data and try to see how to extract emotions from the text information.

(Refer Slide Time: 00:39)



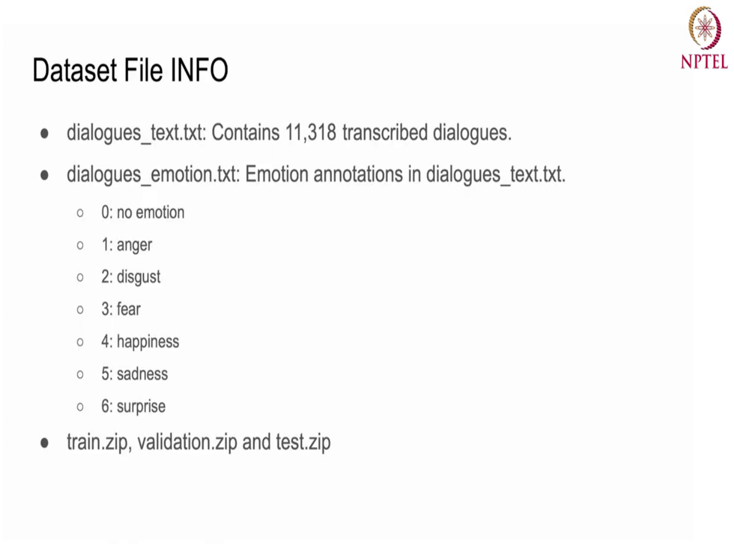
Dataset

- DailyDialog: A Manually Labelled Multi-turn Dialogue Dataset
  - Dialogues in the dataset reflect daily communication
- Download LINK: http://yanran.li/dailydialog
- Licensed under CC BY-NC-SA 4.0.
- Contains 13,118 multi-turn dialogues
- Average Speaker Turns Per Dialogue = 7.9

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. DailyDialog: A Manually Labelled Multi-turn Dialogue Dataset. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 986–995, Taipei, Taiwan. Asian Federation of Natural Language Processing

So, directly starting with the dataset, for this tutorial, we will be using this daily dialog dataset, which is essentially a manually labelled multi-turn dialogue dataset. So, this dataset contains basically 13,118 multi-turn dialogues and dialogues in this dataset reflect daily life communications. So, one can easily download this dataset from this link and this data is published under creative common license.

(Refer Slide Time: 01:09)



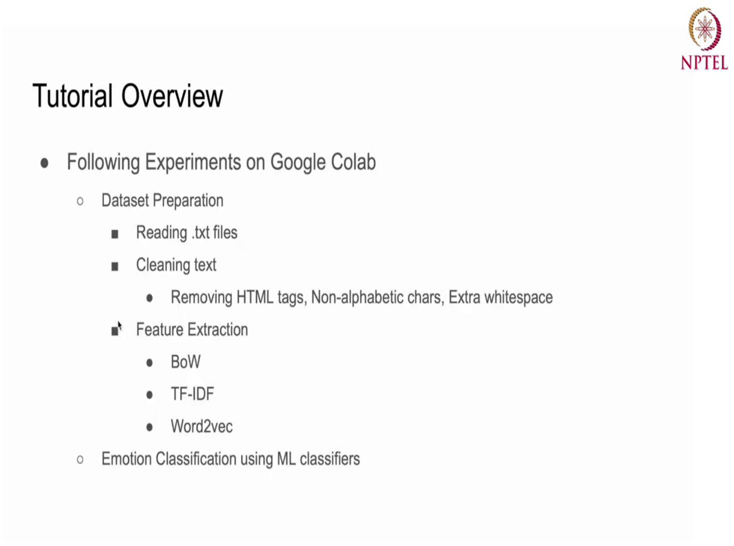
Dataset File INFO

- dialogues_text.txt: Contains 11,318 transcribed dialogues.
- dialogues_emotion.txt: Emotion annotations in dialogues_text.txt.
  - 0: no emotion
  - 1: anger
  - 2: disgust
  - 3: fear
  - 4: happiness
  - 5: sadness
  - 6: surprise
- train.zip, validation.zip and test.zip

So, talking about the dataset file information, dataset will contain a couple of files. In these files, we will be basically interested in dialog underscore text file, which essentially contains these transcribed dialogues. And second file of our interest will be dialogue underscore emotion dot text, which contains the emotion and rotation in the original dialogue underscore text file.

Dialogue annotation will look something like number from 0 to 6, where number 0 will be representing no emotion, number 1 will be representing anger, number 2 will represent disgust, and 3 as a fear, 4 happiness, 5 sadness and 6 surprise.

(Refer Slide Time: 01:55)



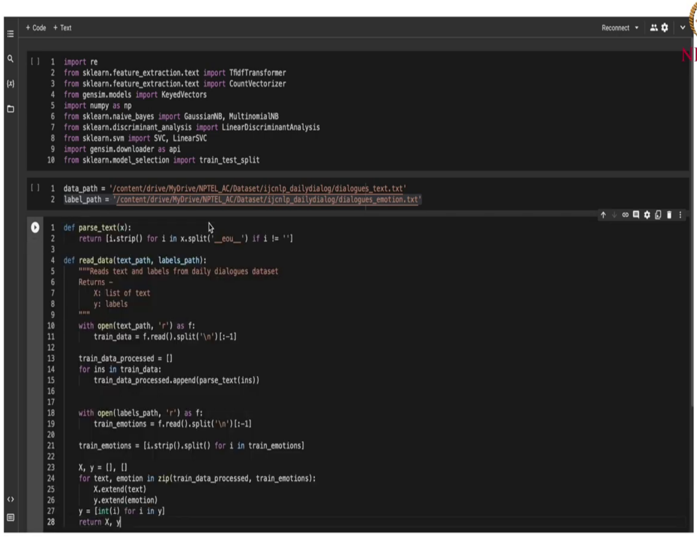
Tutorial Overview

- Following Experiments on Google Colab
  - Dataset Preparation
    - Reading .txt files
    - Cleaning text
      - Removing HTML tags, Non-alphabetic chars, Extra whitespace
    - Feature Extraction
      - BoW
      - TF-IDF
      - Word2vec
  - Emotion Classification using ML classifiers

Now, let me give you a very a brief overview of this tutorial. In this tutorial, we will be performing following experiment on Google Colab. First, we will start with data preparation, where we will be reading text file from Google Drive and we will be cleaning these text files in terms of removing HTML tags, removing non-alphabetic characters, extra white space and removing stop words.

Later, we will be using these common feature extraction method from x data, namely, bag of word, TF-IDF and Word2vec (Refer Time: 02:34) And after extracting these particular features, we will be performing our emotion classification using machine learning classifiers.

(Refer Slide Time: 02:45)



```python
import re
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from gensim.models import KeyedVectors
import numpy as np
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC, LinearSVC
import gensim.downloader as api
from sklearn.model_selection import train_test_split
```

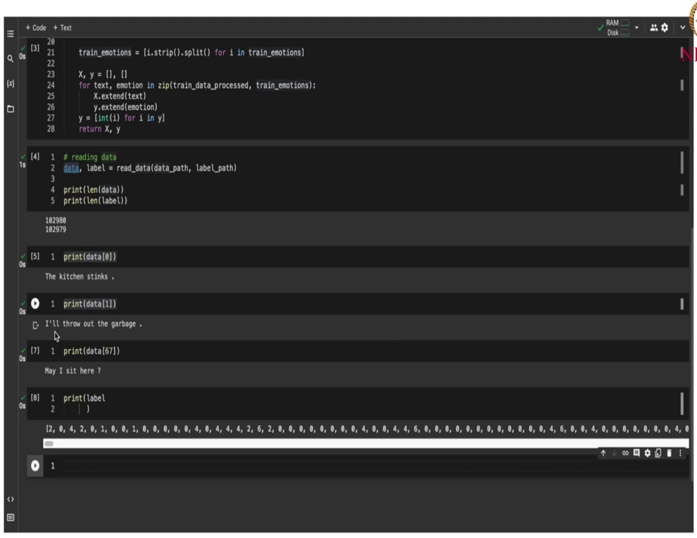
```python
data_path = '/content/drive/MyDrive/NPTEL_AC/Dataset/ijcnlp_dailydialog/dialogues_text.txt'
label_path = '/content/drive/MyDrive/NPTEL_AC/Dataset/ijcnlp_dailydialog/dialogues_emotion.txt'
```

```python
def parse_text(x):
    return [i.strip() for i in x.split('__eou__') if i != '']

def read_data(text_path, labels_path):
    """Reads text and labels from daily dialogues dataset
    Returns -
        X: list of text
        y: labels
    """
    with open(text_path, 'r') as f:
        train_data = f.read().split('\n')[:-1]

    train_data_processed = []
    for ins in train_data:
        train_data_processed.append(parse_text(ins))


    with open(labels_path, 'r') as f:
        train_emotions = f.read().split('\n')[:-1]

    train_emotions = [i.strip().split() for i in train_emotions]

    X, y = [], []
    for text, emotion in zip(train_data_processed, train_emotions):
        X.extend(text)
        y.extend(emotion)
    y = [int(i) for i in y]
    return X, y
```

The coding part, we will start with importing all the essential libraries and the code will look something like this. After importing libraries, we will define our a data and our label path. Code will look something like this. So, after defining our path variable, we will write the code to read all the text files from our Google drive and save them into Python list.

For that, I will write a function which will look something like this. So, you can simply pause this video and try to iterate through each line in this code.

(Refer Slide Time: 03:44)



```
    train_emotions = [i.strip().split() for i in train_emotions]

    X, y = [], []
    for text, emotion in zip(train_data_processed, train_emotions):
        X.extend(text)
        y.extend(emotion)
    y = [int(i) for i in y]
    return X, y
```

```
# reading data
data, label = read_data(data_path, label_path)

print(len(data))
print(len(label))
```

```
102980
102979
```

```
print(data[0])
```

```
The kitchen stinks .
```

```
print(data[1])
```

```
I'll throw out the garbage .
```

```
print(data[67])
```

```
May I sit here ?
```

```
print(label
      )
```

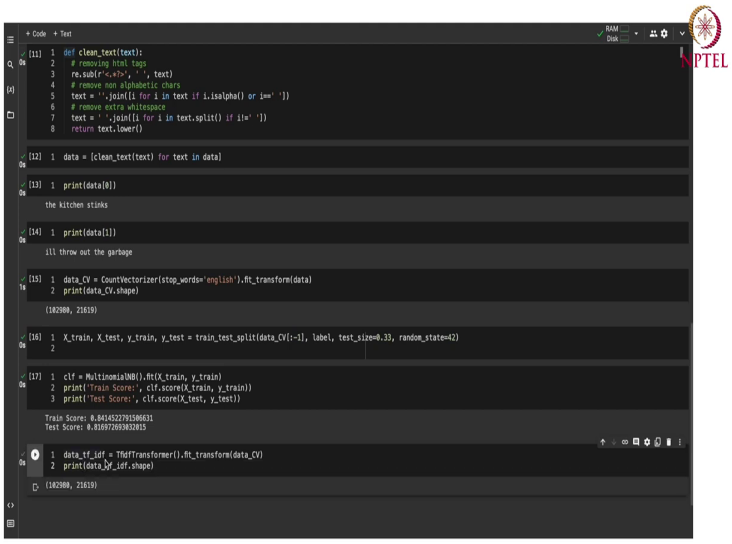
```
[2, 0, 4, 2, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 4, 0, 4, 4, 4, 2, 6, 2, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 4, 4, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 6, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 4, 0
```

And now, we will simply call this function and our data will look something like this. So, we got input data and corresponding labels over here. Maybe I can show you couple of data instances also, so it will look. So, very first line in our data set is the kitchen stinks. Let me show some other line, maybe the second line. So, second line saying, I will throw out the garbage or maybe I can show some random line also that will line add position 67. May sit here ok.

So, he is asking a question over here. So, these line belong to conversation and these line are annotated by annotators into some emotion classes. So, I can show you the emotion classes also. So, here you can see like each line belonging to some sort of a emotion class over here like zero belong to neutral. So, after reading these text files, our first task will be to clean these files.

Most of this text might contain some sort of a noise in terms of punctuation marks, maybe some sort of a hyperlinks. So, before our analysis, we will consider to remove all these sort of a potential noises in our text data. For that, I will write a function and my function will look something like this.

(Refer Slide Time: 05:56)



```
[11]
def clean_text(text):
  # removing html tags
  re.sub(r'<.*?>', ' ', text)
  # remove non alphabetic chars
  text = ''.join([i for i in text if i.isalpha() or i==' '])
  # remove extra whitespace
  text = ' '.join([i for i in text.split() if i!=' '])
  return text.lower()
```

```
[12]
data = [clean_text(text) for text in data]
```

```
[13]
print(data[0])
```

```
the kitchen stinks
```

```
[14]
print(data[1])
```

```
ill throw out the garbage
```

```
[15]
data_CV = CountVectorizer(stop_words='english').fit_transform(data)
print(data_CV.shape)
```

```
(102980, 21619)
```

```
[16]
X_train, X_test, y_train, y_test = train_test_split(data_CV[:-1], label, test_size=0.33, random_state=42)
```

```
[17]
clf = MultinomialNB().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))
```

```
Train Score: 0.8414522791506631
Test Score: 0.816972693032015
```

```
data_tf_idf = TfidfTransformer().fit_transform(data_CV)
print(data_tf_idf.shape)
```

```
(102980, 21619)
```

So, my function is clean text where I will be passing text into this function and it will be trying to remove any sort of a HTML text, then it will remove any sort of a non-alphabetic characters and it will remove any extra white spaces in the text. And in final iteration, it will simply convert my text into lowercase and give it back.

So, let me run this cleaning function on my whole data. So, my data is now clean, so maybe I can show you couple of text instances and try to see the difference between original data and the clean data. So, as we can see here, we have no extra spaces like this and our full stop sign is also removed and all the text is in lower case.

Maybe I will show you (Refer Time: 07:05) also. Here again, you can see like all the punctuation marks are removed and all the text is in lowercase only. So, after performing cleaning operation on our text data, we will start with feature extraction. So, our first feature will be bag of word which we will be implementing using count vectorizer.

So, count vectorizer is basically a tool used in natural language processing to convert words in a document into a numerical representation. And this numerical representation can be easily understand by a machine learning algorithm. It basically counts the number of occurrences of each word and create a table with count for each word in the document. And this table can be used for various tasks are like as text classification, maybe sentiment analysis, topic modelling.

So, to perform our bag of word based feature representation here, we will be using intrinsic function in a scalar library known as count vectorizer. So, you can see the code will look something like this we will use this count vectorizer function and we are also passing argument calls stop word equal to English. So, this argument basically will enable a count vectorizer to remove all the possible stop word present in English language from our text.

And then after removing the stop word, we will make this BOG bag of word representations. If you want to remove some particular sort of stop words, you can also pass a list of words in this argument. So, as we can see count vectorizer has converted the text representation into some particular vector representation and I can see here the dimension of that vector representation.

Now, I can use this representation to learn our machine learning classifier. So, in this case, we will be using, multinomial Naive-Bayes algorithm classify the particular emotional classes. So, before that, we just need to divide our data into respective train and test set. For that, I will be using basic train test split function from a scalar and let us say the test size is 33 percent.

So, we have divided our data into respective train and test sets. Now, I will be using this multinomial, Naive-Bayes classifier and try to see our classification results ok. So, here I can see that we are getting a train score of 84 percent and a test score of 81 percent given that we have a 6 class classification problem and the chance level will come around 16 percent., So, 84 percent train accuracy and 81 percent test accuracy is a good score here.

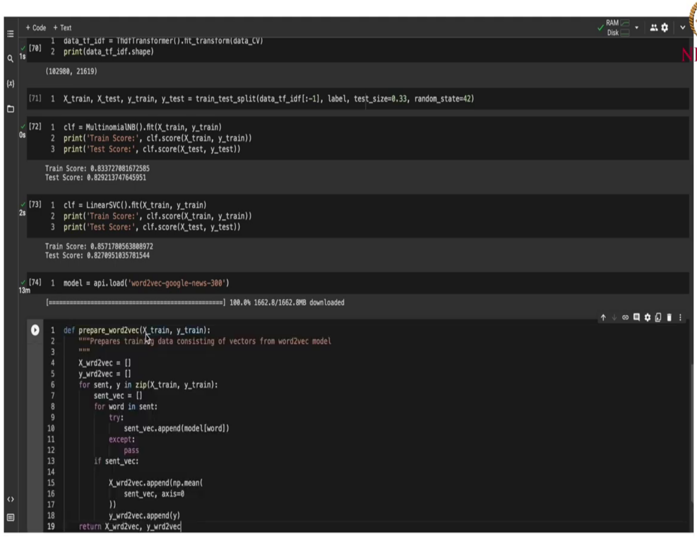
So, after using this count to vectorizer, maybe we can try another sort of feature representation technique known as TF-IDF. So, TF-IDF is basically a; is basically a better version of this count vectorizer. TF-IDF takes into account the importance of word in a document by multiplying the word count by inverse document frequency.

So, what is this inverse document frequency? IDF is basically calculated as logarithm of ratio of total number of documents to the number of documents containing the word. This way, words that are frequent in a particular document, but rare in the corpus as a whole are given more weight weightage. And this approach can help to better capture the meaning and importance of word in a document.

And in general, TF-IDF is considered to be a more advanced and more effective techniques than count vectorizer. So, to use this technique, I will be again using another function from a sklearn library and our code will look something like this ok. Now, we have gotten those TF-IDF representation over here.

So, we can try to train our classifier on these sort of features and try to see, do we get any sort of improvement in term of our emotion classification accuracies. For this, I will again divide my data into train and test splits with TF-IDF as a input over here.

(Refer Slide Time: 12:02)



```
[70]
data_tf_idf = TfidfTransformer().fit_transform(data_CV)
print(data_tf_idf.shape)

(102980, 21619)

[71]
X_train, X_test, y_train, y_test = train_test_split(data_tf_idf[:-1], label, test_size=0.33, random_state=42)

[72]
clf = MultinomialNB().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

Train Score: 0.833727081672585
Test Score: 0.829213747645951

[73]
clf = LinearSVC().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))

Train Score: 0.8571780563808972
Test Score: 0.8270951035781544

[74]
model = api.load('word2vec-google-news-300')

[==================================================] 100.0% 1662.8/1662.8MB downloaded

def prepare_word2vec(X_train, y_train):
    """Prepares training data consisting of vectors from word2vec model
    """
    X_wrd2vec = []
    y_wrd2vec = []
    for sent, y in zip(X_train, y_train):
        sent_vec = []
        for word in sent:
            try:
                sent_vec.append(model[word])
            except:
                pass
        if sent_vec:

            X_wrd2vec.append(np.mean(
                sent_vec, axis=0
            ))
            y_wrd2vec.append(y)
    return X_wrd2vec, y_wrd2vec
```

And I will be reusing my multinomial code and try to see, do I get any sort of better performance here? So, in this case, we can see, we get slightly better test accuracy over here. I will also like to see, this change in a classifier has any sort of effect in our performance.

So, for this, I will be using linear SVC, which is basically a support vector machine with linear kernel. Here, we can see that we get slightly better train score, but test score is somewhat similar. So, after count vectorizer and TF-IDF representations, we will move to our third type of representation, which is called Word2Vec.

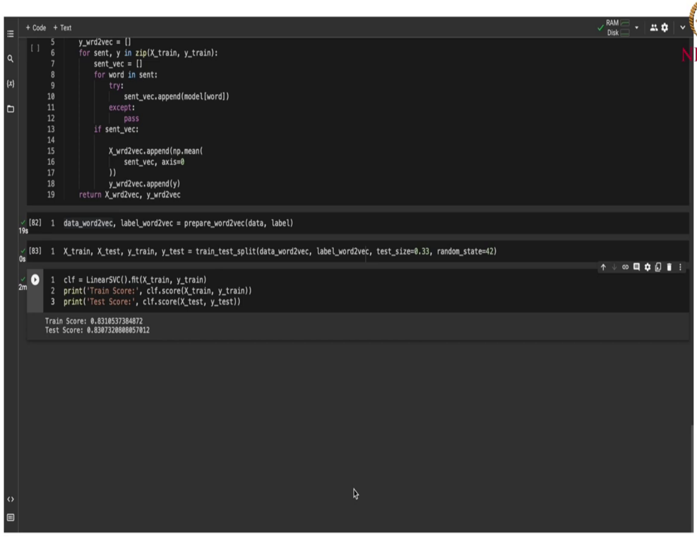
So, Word2Vec is a type of neural network model used for natural language processing. It was developed by researchers at Google in back in 2013. And the purpose of Word2Vec is to create words embedding, which are numerically representation of word that can be used in machine learning models.

This model, I mean, this Word2Vec takes a large corpus of texts input and producer vector for each word in the corpus. These vectors are designed to capture the semantic relationship between the words. So, in our case, we will be using a pre-trained sort of Word2Vec model.

So, we will download this pre-trained model using Jensen API and the code will look something like this. So, downloading this code may take a good amount of time because this model will be a little bit larger model. So, please have some patience. So, after downloading our model, we will write a function a name as prepare Word2Vec.

So, this function will basically prepare training data consisting of vectors from Word2Vec model and the code will look something like this. And after that, I will simply create my Word2Vec representation by calling this function.

(Refer Slide Time: 14:40)



```
    y_wrd2vec = []
    for sent, y in zip(X_train, y_train):
        sent_vec = []
        for word in sent:
            try:
                sent_vec.append(model[word])
            except:
                pass
        if sent_vec:

            X_wrd2vec.append(np.mean(
                sent_vec, axis=0
            ))
            y_wrd2vec.append(y)
    return X_wrd2vec, y_wrd2vec
```

```
[82] data_word2vec, label_word2vec = prepare_word2vec(data, label)
```

```
[83] X_train, X_test, y_train, y_test = train_test_split(data_word2vec, label_word2vec, test_size=0.33, random_state=42)
```

```
clf = LinearSVC().fit(X_train, y_train)
print('Train Score:', clf.score(X_train, y_train))
print('Test Score:', clf.score(X_test, y_test))
```

```
Train Score: 0.8310537384872
Test Score: 0.8307320808057012
```

And this code also might take the little bit of time ok. Now, my Word2Vec representations are created. I will simply use my train test split over these Word2Vec representations and later I will simply learn my machine learning classifier. And the code will look something like this.

Let us see how linear SVM classify our motion classes using these Word2Vec representations. So, as we can see this Word2Vec representations give us a very nice code we are using linear SVC. So, concluding this tutorial, in this tutorial, we explored a public dataset for classifying different sort of emotions.

We started with cleaning text data by removing HTML tags, non-alphabetic vectors and extra-white spaces. Then we used our three different sort of feature extraction method, bag of word, TF-IDF and Word2Vec and we used machine learning based classifier for classifying these emotion classes.