

Exercise- Amazon Fine Food Reviews LSTM Model.

Amazon Fine Food Reviews Analysis

Data Source: <https://www.kaggle.com/snap/amazon-fine-food-reviews>

EDA: <https://nycdatasience.com/blog/student-works/amazon-fine-foods-visualization/>

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454

Number of users: 256,059

Number of products: 74,258

Timespan: Oct 1999 - Oct 2012

Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unique identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2) Getting Vocabulary of all the words Getting Frequency of each word Indexing Each word Convert your data into imdb dataset format Run the Istm model and Report the Accuracy

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

```
from google.colab import drive
drive.mount("/content/drive")
```

➞ Drive already mounted at /content/drive; to attempt to forcibly remount, call dri

```
# Code to read csv file into colaboratory:
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
```

```

from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

#2. Get the file
downloaded = drive.CreateFile({'id': '1il5Ue_HoshloviEApbr05ncK729yeUZY'}) # replace the
downloaded.GetContentFile('database.sqlite')

```

▼ [1]. Reading Data

▼ [1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation will be set to "positive". Otherwise, it will be set to "negative".

```

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords

from tqdm import tqdm
import os

# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

```

```
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

➤ Number of data points in our data (525814, 10)

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
print(display.shape)
display.head()
```


➤ (80668, 7)

	UserId	ProductId	ProfileName	Time	Score	Text
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "happy"	1342396800	5	My wife has recurring extreme muscle

```
display[display['UserId']=='AZY10LLTJ71NX']
```

	UserId	ProductId	ProfileName	Time	Score	Text
						I wa undertheshrine recommende

```
display['COUNT(*)'].sum()
```


 393063

▼ **[2] Exploratory Data Analysis**

▼ **[2.1] Data Cleaning: Deduplication**

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```



	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delete the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False)
```

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId", "ProfileName", "Time", "Text"}, keep='first')
final.shape
```

```
(364173, 10)
```

```
#Sorting data according to Time in ascending order
final=final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort',
```

```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
69.25890143662969
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calculations

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)
display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)
```

```
#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(364171, 10)
1      307061
0       57110
Name: Score, dtype: int64
```

▼ Taking first 60k Rows

```
# This is formatted as code
```

```
# We will collect first 60000 rows without repetition from time_sorted_data dataframe
final = final[:60000]
print(final.shape)
final.head()
```

```
(60000, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1

▼ [3] Preprocessing

Tokenization basically refers to splitting up a larger body of text into smaller lines, words or even creating words for a non-English language.

A sentence or data can be split into words using the method `word_tokenize()`:

Tokenizer to vectorize the text and convert it into sequence of integers after restricting the tokenizer to use only top most common 2500 words. I used `pad_sequences` to convert the sequences into 2-D numpy array.

```
final['Text'].isnull().values.any()
final['Score'].isnull().values.any()
```

False

```
# Calculating number of words in each Review
final['word_count'] = final['Text'].apply(lambda x: len(str(x).split(" ")))
final[['Text', 'word_count']].head()
```

Text word_count

138706	this witty little book makes my son laugh at l...	79
138683	I can remember seeing the show when it aired o...	84
417839	Beetlejuice is a well written movie ever...	29
346055	A twist of rumplestiskin captured on film, sta...	38
417838	Beetlejuice is an excellent and funny movie. K...	44

```
# Calculating Number of characters
final['char_count'] = final['Text'].str.len() ## this also includes spaces
final[['Text', 'char_count']].head()
```

Text char_count

138706	this witty little book makes my son laugh at l...	375
138683	I can remember seeing the show when it aired o...	407
417839	Beetlejuice is a well written movie ever...	166
346055	A twist of rumplestiskin captured on film, sta...	222
417838	Beetlejuice is an excellent and funny movie. K...	244

```
# Calculating Average Word Length
def avg_word(sentence):
    words = sentence.split()
    return (sum(len(word) for word in words)/len(words))

final['avg_word'] = final['Text'].apply(lambda x: avg_word(x))
final[['Text', 'avg_word']].head()
final[['Text', 'avg_word']].max()
```

Text ~Earth's Best Infant Formula Soy Iron, 13.2-Ou...
avg_word 12.9259
dtype: object

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can't", "can not", phrase)

    # general
    phrase = re.sub(r"n't", " not", phrase)
    phrase = re.sub(r"\ 're", " are", phrase)
    phrase = re.sub(r"\ 's", " is", phrase)
```

```

phrase = re.sub(r"\d", " would", phrase)
phrase = re.sub(r"\ll", " will", phrase)
phrase = re.sub(r"\t", " not", phrase)
phrase = re.sub(r"\ve", " have", phrase)
phrase = re.sub(r"\m", " am", phrase)
return phrase

```

```

from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence)
    sentence = re.sub('[^A-Za-z0-9]+', ' ', sentence)
    sentence = re.sub(r"http\S+", "", sentence)
    preprocessed_reviews.append(sentence.strip())

```

```

↳ 100%|██████████| 60000/60000 [00:04<00:00, 12676.43it/s]

```

```

print(type(preprocessed_reviews))
print(type(final['Score']))
print(len(preprocessed_reviews))
print(len(final['Score'])).

```

```

↳ <class 'list'>
   <class 'pandas.core.series.Series'>
   60000
   60000

```

▼ Getting Vocabulary of all the words

```

from keras.preprocessing.text import Tokenizer
# create the tokenizer
t = Tokenizer(filters='!#$%&()*+,-./:;<=>?@[\\]^_`{|}~ ', lower=True,)
# fit the tokenizer on the documents
t.fit_on_texts(preprocessed_reviews)
sequences = t.texts_to_sequences(preprocessed_reviews)

```

```

↳ Using TensorFlow backend.

```

▼ Getting Frequency of each word

```

# summarize what was learned
# word_counts: A dictionary of words and their counts.
word_counts=t.word_counts
print('Found %s unique tokens.' % len(word_counts))
print(word_counts)
word_index=t.word_index
print('Found %s unique tokens.' % len(word_index))
print(word_index)
#document_count:An integer count of the total number of documents that were used to fit
document_count=t.document_count
print('Found %s unique tokens.' % (document_count))
print(document_count)
#word_docs: A dictionary of words and how many documents each appeared in.
word_docs=t.word_docs
print('Found %s unique tokens.' %len(word_docs))
print(word_docs)

```

```

↳

```



```

Found 43840 unique tokens.
OrderedDict([('this', 64470), ('witty', 5), ('little', 7953), ('book', 352), ('ma
Found 43840 unique tokens.
{'the': 1, 'i': 2, 'and': 3, 'a': 4, 'is': 5, 'it': 6, 'to': 7, 'of': 8, 'br': 9,
Found 60000 unique tokens.
60000
Found 43840 unique tokens.
defaultdict(<class 'int'>. {'wittv': 5, 'little': 6627, 'to': 3952, 'will': 1277

```

▼ Indexing Each word

```

print(type(word_docs)).
word_docs_sorted = sorted(word_docs.items(), key = lambda x: (x[1],x[0]))
print(word_docs_sorted)

```

```

↳ <class 'collections.defaultdict'>
[('aaaa', 1), ('aaaaaa', 1), ('aaaaaaaaaaaa', 1), ('aaaaaaaaaaaaaaaa', 1), ('aaaa

```

▼ Splitting Data into Train and Test

```

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(sequences ,final['Score'], test_size
print(x_train)
print(x_test)
print(y_train)
print(y_test)

```

↳

```

# Importing libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import Dropout
# fix random seed for reproducibility
np.random.seed(7).

```

▼ Padding

```

# truncate and/or pad input sequences
max_review_length = 400
X_train = sequence.pad_sequences(x_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(x_test, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])

```

↳

▼ LSTM with 1-layer

```

%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()

top_words = 5000
# create the model
embedding_vecor_length = 32

# Initialising the model
model_1 = Sequential()

# Adding embedding
model_1.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))

# Adding Dropout
model_1.add(Dropout(0.2))

# Adding first LSTM layer
#The first layer is an LSTM layer with 300 memory units and it returns sequences
model_1.add(LSTM(100))

# Adding Dropout
model_1.add(Dropout(0.2))

# Adding output layer
model_1.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_1.summary())

# Compiling the model
model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_1 = model_1.fit(X_train, y_train, nb_epoch=10, batch_size=512, verbose=1, validate

```



```

# Final evaluation of the model
scores = model_1.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: %.2f%%" % (scores[1]*100))

# Final evaluation of the model
scores = model_1.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: %.2f%%" % (scores[1]*100))

# Test and Train accuracy of the model
model_1_test = scores[1]
model_1_train = max(history_1.history['acc'])

```



Train Accuracy: 97.81%
Test Accuracy: 92.76%

▼ Train and Test Loss

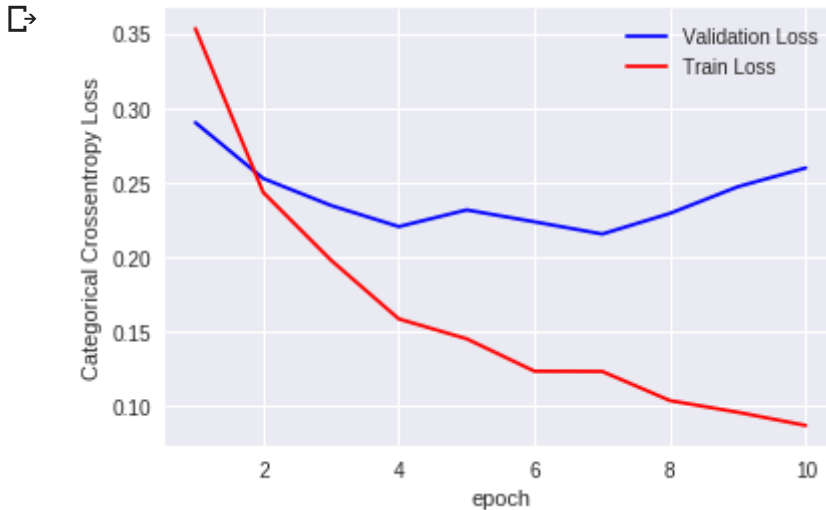
```
%matplotlib inline

import matplotlib.pyplot as plt

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1, 11))

vy = history_1.history['val_loss']
ty = history_1.history['loss']
plt.grid()
plt_dynamic(x, vy, ty, ax)
```



▼ LSTM with 2Hidden layers

```
# create the model
embedding_vecor_length = 32

# Initialising the model
model_2 = Sequential()

# Adding embedding
model_2.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))

# Adding first LSTM layer
model_2.add(LSTM(100,return_sequences=True, dropout=0.4, recurrent_dropout=0.4))

# Adding second LSTM layer
model_2.add(LSTM(100, dropout=0.4, recurrent_dropout=0.4))

# Adding output layer
model_2.add(Dense(1, activation='sigmoid'))

# Printing the model summary
print(model_2.summary())

# Compiling the model
model_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_2 = model_2.fit(X_train, y_train, nb_epoch=10, batch_size=512, verbose=1, validat
```



Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 400, 32)	160000
lstm_2 (LSTM)	(None, 400, 100)	53200
lstm_3 (LSTM)	(None, 100)	80400
dense_2 (Dense)	(None, 1)	101
Total params: 293,701		
Trainable params: 293,701		
Non-trainable params: 0		

None

Train on 42000 samples, validate on 18000 samples

Epoch 1/10

42000/42000 [=====] - 181s 4ms/step - loss: 0.3529 - acc

Epoch 2/10

42000/42000 [=====] - 180s 4ms/step - loss: 0.2735 - acc

Epoch 3/10

42000/42000 [=====] - 177s 4ms/step - loss: 0.2543 - acc

Epoch 4/10

42000/42000 [=====] - 180s 4ms/step - loss: 0.2621 - acc

Epoch 5/10

42000/42000 [=====] - 180s 4ms/step - loss: 0.2596 - acc

Epoch 6/10

42000/42000 [=====] - 180s 4ms/step - loss: 0.2676 - acc

Epoch 7/10

42000/42000 [=====] - 180s 4ms/step - loss: 0.2706 - acc

Epoch 8/10

42000/42000 [=====] - 180s 4ms/step - loss: 0.2725 - acc

Epoch 9/10

Final evaluation of the model

scores = model_2.evaluate(X_train, y_train, verbose=0)

print("Train Accuracy: %.2f%%" % (scores[1]*100))

Final evaluation of the model

scores = model_2.evaluate(X_test, y_test, verbose=0)

print("Test Accuracy: %.2f%%" % (scores[1]*100))

Test and train accuracy of the model

model_2_test = scores[1]

model_2_train = max(history_2.history['acc'])

☞ Train Accuracy: 90.45%

Test Accuracy: 88.95%

fig,ax = plt.subplots(1,1)

ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

x = list(range(1,11))

Validation loss

vy = history_2.history['val_loss']

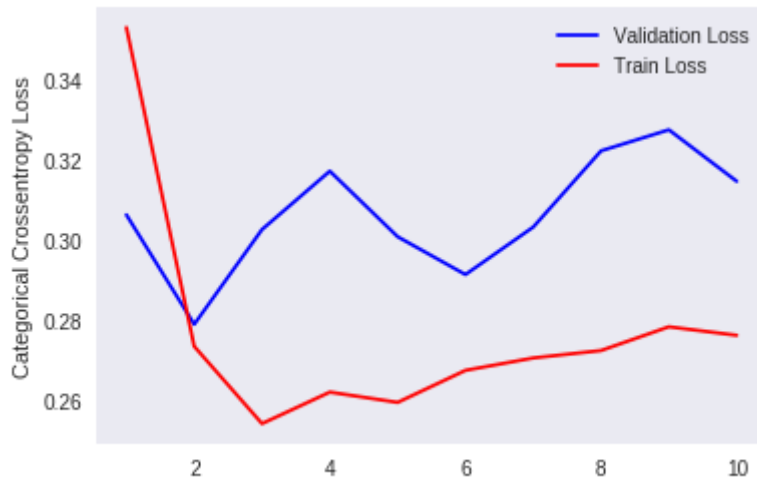
Training loss

ty = history_2.history['loss']

Calling the function to draw the plot

plt_dynamic(x, vy, ty,ax)

☞



▼ LSTM With 3-Hidden layers

```
# create the model
embedding_vecor_length = 32

# Initialising the model
model_3 = Sequential()

# Adding embedding
model_3.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))

# Adding first LSTM layer
model_3.add(LSTM(100,return_sequences=True, dropout=0.4, recurrent_dropout=0.4))

# Adding second LSTM layer
model_3.add(LSTM(100,return_sequences=True, dropout=0.5, recurrent_dropout=0.5))

# Adding third LSTM layer
model_3.add(LSTM(100, dropout=0.4, recurrent_dropout=0.4))

# Adding output layer
model_3.add(Dense(1, activation='relu'))

# Printing the model summary
print(model_3.summary())

# Compiling the model
model_3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fitting the data to the model
history_3 = model_3.fit(X_train, y_train, nb_epoch=10, batch_size=1024 ,verbose=1,valida
```



Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 400, 32)	160000
lstm_4 (LSTM)	(None, 400, 100)	53200
lstm_5 (LSTM)	(None, 400, 100)	80400
lstm_6 (LSTM)	(None, 100)	80400
dense_3 (Dense)	(None, 1)	101
Total params: 374,101		
Trainable params: 374,101		
Non-trainable params: 0		

None

Train on 42000 samples, validate on 18000 samples

Epoch 1/10

42000/42000 [=====] - 194s 5ms/step - loss: 1.9541 - acc

Epoch 2/10

42000/42000 [=====] - 190s 5ms/step - loss: 1.2771 - acc

Final evaluation of the model

```
scores = model_3.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: %.2f%%" % (scores[1]*100))
```

Final evaluation of the model

```
scores = model_3.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: %.2f%%" % (scores[1]*100))
```

Test and train accuracy of the model

```
model_3_test = scores[1]
model_3_train = max(history_3.history['acc'])
```

☞ Train Accuracy: 88.99%
Test Accuracy: 87.61%

Epoch 10/10

Plotting Train and Test Loss VS no. of epochs

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,11))
```

Validation loss

```
vy = history_3.history['val_loss']
```

Training loss

```
ty = history_3.history['loss']
```

Calling the function to draw the plot

```
plt_dynamic(x, vy, ty,ax).
```

☞



▼ CONCLUSIONS

```
from prettytable import PrettyTable
print("\n Model performance table of LSTM")
x = PrettyTable()
x.field_names=["Model","Training Accuracy","Test Accuracy"]

x.add_row(["RNN With 1 LSTM Layer", 97.81,92.76,])
x.add_row(["RNN With 2 LSTM Layer",90.45 ,88.95])
x.add_row(["RNN With 3 LSTM Layer",88.99 ,87.61])
print(x)
```



Model performance table of LSTM

Model	Training Accuracy	Test Accuracy
RNN With 1 LSTM Layer	97.81	92.76
RNN With 2 LSTM Layer	90.45	88.95
RNN With 3 LSTM Layer	88.99	87.61

▼ Steps Involved:

- 1) Connecting SQL file
- 2) Data Cleaning
- 3) Sorting the data based on time
- 4) Taking 1st 60K Rows
- 5) Data Preprocessing
- 6) Tokenization
- 7) Getting Vocabulary of each word
- 8) Getting Frequency of each word
- 9) Index of each word
- 10) Splitting data into train and test based on time (70:30)
- 11) padding
- 12) Running LSTM model
- 13) Conclusion

