AMITY UNIVERSITY MADHYA PRADESH

LAB MANUAL

on

TEXT AND SOCIAL MEDIA ANALYTICS LAB

CSD – 521

PREPARED BY

Dr. HEMANT KUMAR SONI

Department of Computer Science & Engineering
Amity School of Engineering & Technology
AMITY UNIVERSITY MADHYA PRADESH,GWALIOR
JULY 2025

TEXT AND SOCIAL MEDIA ANALYTICS LAB

Course Code: CSD 521 **Credit Unit: 01**
Total Hours: 20

Course Objective :
The objective of the course is to provide the understanding of the fundamental graphical operations and the implementation on computer, the mathematics behind computer graphics, including the use of spline curves and surfaces. It gives the glimpse of recent advances in computer graphics, user interface issues that make the computer easy, for the novice to use.

SOFTWARE REQUIREMENTS: Python 3.6, Anaconda IDE with Spider

List of experiments/demonstrations:

1. Write python code to flatten and evaluate a deep tree in NLP
2. Create Shallow Tree in NLP and print its height
3. Download wine quality data set from the UCI Machine Learning Repository which is available for free. Then print data of five rows of red and white wines. Check for NULL Values in red wine. Create a histogram to show distribution of alcohol and finally split the data for training and validation.
4. Avengers Endgame and Deep learning. Write python code to implement Image Caption Generation using the Avengers End Games Characters
5. Create a Neural network using Python (you can use NumPy to implement this)
6. Implement Word Embedding using Word2Vec
7. Collocations are two or more words that tend to appear frequently together, for example – United States. Implement this using Python.
8. WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing. Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept Show working of these using Python
9. Implement Naïve Baye's Classifier using python.
10. Twitter Sentiment Analysis using Python. Fetch tweets from twitter using Python and implement it.

Examination Scheme:

IA			EE			
A	**PR**	**Practical Based Test**	**Major Experiment**	**Minor Experiment**	**LR**	**Viva**
5	10	15	35	15	10	10

Note: IA –Internal Assessment, EE- External Exam, A- Attendance PR- Performance, LR – Lab Record, V – Viva.

Course Outcomes:

After taking this course, you will be able to:

- Utilize various Application Programming Interface (API) services to collect data from different social media sources such as YouTube, Twitter, and Flickr.
- Process the collected data - primarily structured - using methods involving correlation, regression, and classification to derive insights about the sources and people who generated that data.
- Analyze unstructured data - primarily textual comments - for sentiments expressed in them.
- Use different tools for collecting, analyzing, and exploring social media data for research and development purposes.

Text & References*:*

Text:

- Mining Text Data. Charu C. Aggarwal and ChengXiang Zhai, Springer, 2012.
- Speech & Language Processing. Dan Jurafsky and James H Martin, Pearson Education India, 2000.

References:

- Introduction to Information Retrieval. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schuetze, Cambridge University Press, 2007.

LIST OF EXPRIMENTS

1. Write python code to flatten and evaluate a deep tree in NLP
2. Create Shallow Tree in NLP and print its height
3. Download wine quality data set from the UCI Machine Learning Repository which is available for free. Then print data of five rows of red and white wines. Check for NULL Values in red wine. Create a histogram to show distribution of alcohol and finally split the data for training and validation.
4. Avengers Endgame and Deep learning. Write python code to implement Image Caption Generation using the Avengers End Games Characters
5. Create a Neural network using Python (you can use NumPy to implement this)
6. Implement Word Embedding using Word2Vec
7. Collocations are two or more words that tend to appear frequently together, for example – United States. Implement this using Python.
8. WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing. Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept Show working of these using Python
9. Implement Naïve Baye's Classifier using python.
10. Twitter Sentiment Analysis using Python. Fetch tweets from twitter using Python and implement it.

1. Write python code to flatten and evaluate a deep tree in NLP

Part #1 : Class for flattening the deep tree

```
from nltk.tree import Tree

def flatten_childtrees(trees):
    children = []

    for t in trees:
        if t.height() < 3:
            children.extend(t.pos())

        elif t.height() == 3:
            children.append(Tree(t.label(), t.pos()))

        else:
            children.extend(
                flatten_childtrees([c for c in t]))

    return children


def flatten_deeptree(tree):
    return Tree(tree.label(),
                flatten_childtrees([c for c in tree ]))
```

Part #2 : Evaluating `flatten_deeptree()`

```
!pip install transforms
import nltk
nltk.download('treebank')

from nltk.corpus import treebank
from transforms import flatten_deeptree

print ("Deep Tree : \n", treebank.parsed_sents()[0])

print ("\nFlattened Tree : \n",
    flatten_deeptree(treebank.parsed_sents()[0]))
```

Output :

```
Deep Tree :
 (S
```

```
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (,, )
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (,, ))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))
```

```
Flattened Tree :
Tree('S', [Tree('NP', [('Pierre', 'NNP'), ('Vinken', 'NNP')]), (',
',
', '), Tree('NP', [('61', 'CD'), ('years', 'NNS')]), ('old', 'JJ'),
(', ', ', '), ('will', 'MD'), ('join', 'VB'), Tree('NP', [('the',
'DT'), ('board', 'NN')]), ('as', 'IN'), Tree('NP', [('a', 'DT'),
('nonexecutive', 'JJ'), ('director', 'NN')]), Tree('NP-TMP',
[('Nov.',
'NNP'), ('29', 'CD')]), ('.', '.')])
```

How it works ?

flatten_deeptree() : returns a new Tree from the given tree by calling flatten_childtrees() on each of the given tree's children.

flatten_childtrees() : Recursively drills down into the Tree until it finds child trees whose height() is equal to or less than 3.

Part #3 : height()

from nltk.corpus import treebank

from transforms import flatten_deeptree

```
from nltk.tree import Tree


print ("Height : ",
        Tree('NNP', ['Pierre']).height())


print ("\nHeight : ", Tree(
                'NP', [Tree('NNP', ['Pierre']),
                                        Tree('NNP', ['Vinken'])]). height())
```

Output :

```
Height : 2


Height : 3
```

2. Create Shallow Tree in NLP and print its height

Part #1 : Lets' understand `shallow_tree()`

```
from nltk.tree import Tree


def shallow_tree(tree):
        children = []


    for t in tree:
        if t.height() < 3:
            children.extend(t.pos())
    else:
        children.append(Tree(t.label(), t.pos()))


    return Tree(tree.label(), children)
```

Part #2 : Evaluating

```
!pip install scrapy

import scrapy

from scrapy.linkextractors import LinkExtractor

#import transforms


#from transforms import shallow_tree

#shallow_tree(treebank.parsed_sents()[0])

from nltk.corpus import treebank


print ("Deep Tree : \n", treebank.parsed_sents()[0])


print ("\nShallow Tree : \n", shallow_tree(treebank.parsed_sents()[0]) )
```

Output :

```
Deep Tree :
 (S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (,, )
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (,, ))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))


Shallow Tree :
Tree('S', [Tree('NP-SBJ', [('Pierre', 'NNP'), ('Vinken', 'NNP'),
(', ', ', '),
('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (', ', ', ')]),
Tree('VP', [('will', 'MD'), ('join', 'VB'), ('the', 'DT'),
('board', 'NN'),
('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director',
'NN'),
('Nov.', 'NNP'), ('29', 'CD')]), ('.', '.')])
```

How it works ?

- shallow_tree() function creates new child trees by iterating over each of the top-level subtrees.
- The subtree is replaced by a list of its part-of-speech tagged children, if the height() of a subtree is less than 3.
- If children of a tree are the part-of-speech tagged leaves, the All other subtrees are replaced by a new Tree.
- Thus, eliminates all the nested subtrees while still retaining the top-level subtrees.

Part #3 : height

```
print ("height of tree : ",
```

```
        treebank.parsed_sents()[0].height())

print ("\nheight of shallow tree : ",
        shallow_tree(treebank.parsed_sents()[0]).height())
```

Output :

```
height of tree : 7


height of shallow tree :3
```

3. Download wine quality data set from the UCI Machine Learning Repository which is available for free. Then print data of five rows of red and white wines. Check for NULL Values in red wine. Create a histogram to show distribution of alcohol and finally split the data for training and validation.

Part #1 : Get the data and convert them to a format of a data frame.

```
# Load in the red wine data from the UCI ML website.

df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv',sep=';')

# Take a look

print(df.head(10))

# Data dimensionality (rows, colums)

print(df.shape)

# Data distributing

df.info()
```

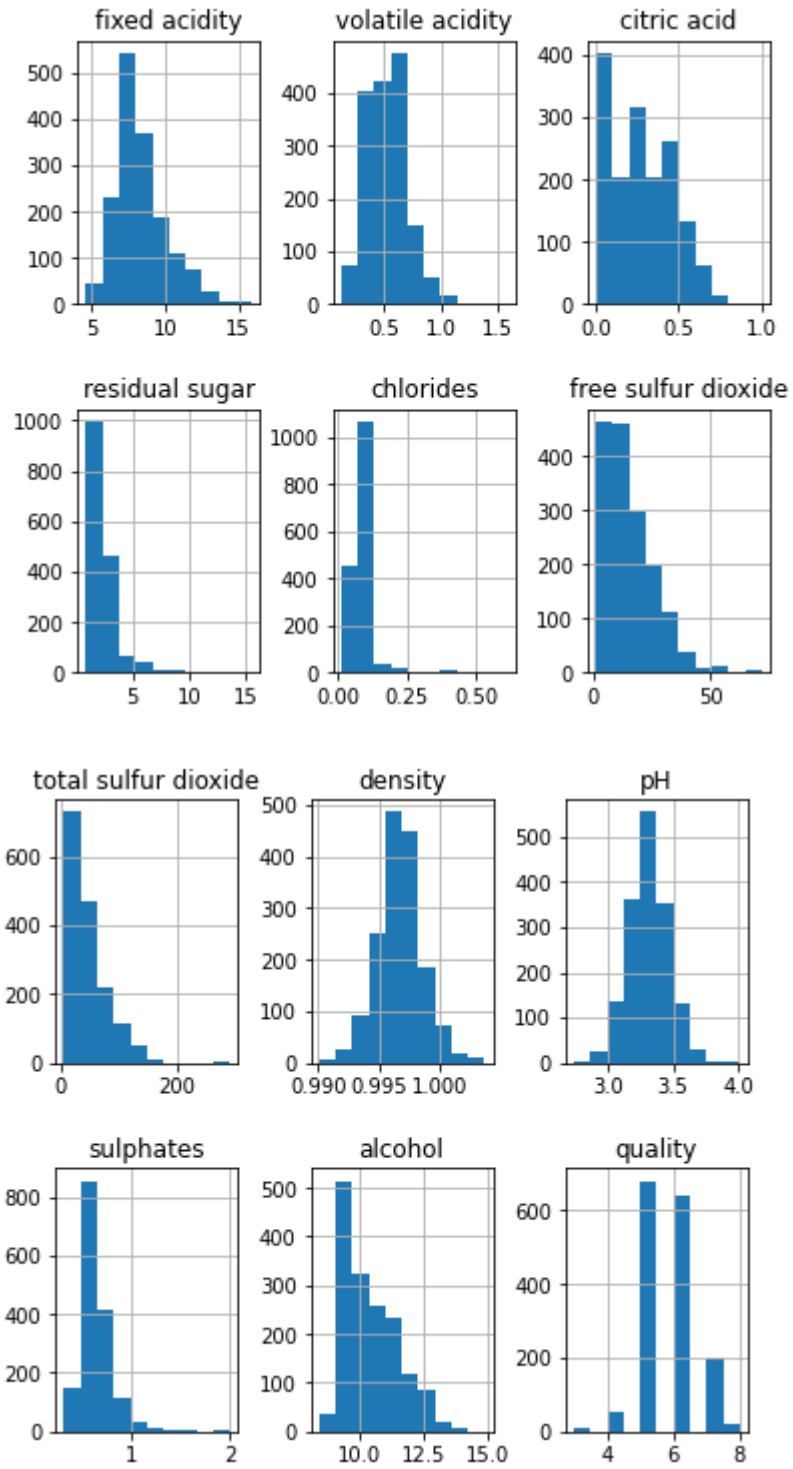
Part#2 : Data Splitting.

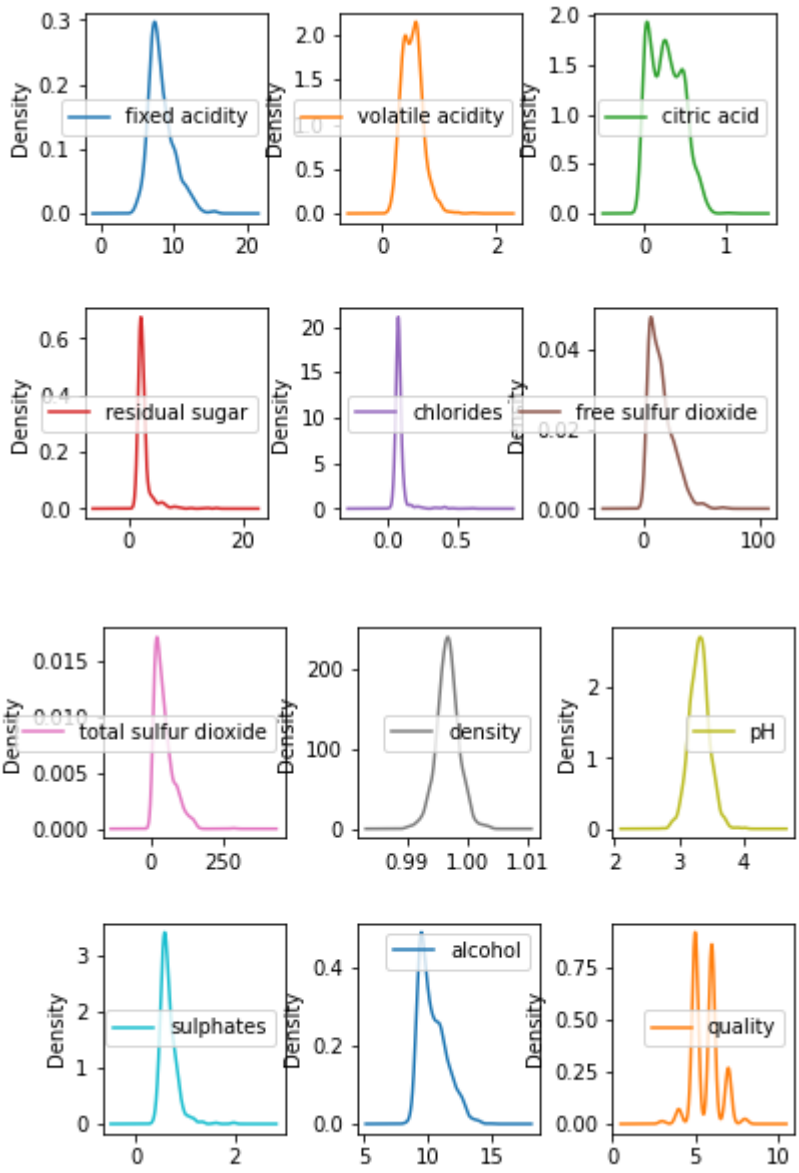
The data has been split into two groups: training set 80%, test set 20%.

```
# Now seperate the dataset as response variable and feature variabes

X = df.drop('quality', axis=1)

y = df['quality']

# Train and Test splitting of data

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)

# Applying Standard scaling to get optimized result

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.fit_transform(X_test)
```

Part#4 : Visualizing data

```
# Histograms
import matplotlib.pyplot as plt
df.hist(bins=10,figsize=(6, 5))
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=2.0,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
# Density
df.plot(kind='density', subplots=True, layout=(4,3), sharex=False)
plt.subplots_adjust(left=0.1,
                    bottom=0.1,
                    right=0.9,
                    top=2.0,
                    wspace=0.4,
                    hspace=0.4)
plt.show()
```


fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality


fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
quality
Density

4. Avengers Endgame and Deep learning. Write python code to implement Image Caption Generation using the Avengers End Games Characters

Prerequisites-

Anaconda

Pytorch

MSCOCO Dataset

Dataset

git clone https://github.com/pdollar/coco.git

cd coco/PythonAPI/

make

python setup.py build

python setup.py install

cd ../../

git clone https://github.com/yunjey/pytorch-tutorial.git

cd pytorch-tutorial/tutorials/03-advanced/image_captioning/

pip install -r requirements.txt

Pretrained model –

Let's download the pretrained model and the vocabulary file from here, then we should extract pretrained_model.zip to ./models/ and vocab.pkl to ./data/ using the unzip command.

Now the model is ready which can predict the captions using:

$ python sample.py --image='/example.png'

Import all the libraries and make sure the notebook is in the root folder of the repository:

import torch

import matplotlib.pyplot as plt

import numpy as np

import argparse

```
import pickle

import os

from torchvision import transforms

from PIL import Image


# this file is located in pytorch tutorial/image

# captioning which we pull from git remember

from build_vocab import Vocabulary

from model import EncoderCNN, DecoderRNN

# Model path


# make sure path must correct

ENCODER_PATH = 'content/encoder-5-3000.pkl'

DECODER_PATH = 'content/decoder-5-3000.pkl'

VOCAB_PATH = 'content/vocab.pkl'


# CONSTANTS because of architecture what we are using

EMBED_SIZE = 256

HIDDEN_SIZE = 512

NUM_LAYERS = 1
```

To load_image Add this configuration code:

```
# Device configuration snippet

device = torch.cuda.device(0) # 0 represent default device


# Function to Load and Resize the image

def load_image(image_path, transform=None):

image = Image.open(image_path)
```

```
image = image.resize([224, 224], Image.LANCZOS)
if transform is not None:
        image = transform(image).unsqueeze(0)
return image
```

Now, let's code a PyTorch function which uses pretrained data files to predict the output:

```
def PretrainedResNet(image_path, encoder_path=ENCODER_PATH,
                                        decoder_path=DECODER_PATH,
                                        vocab_path=VOCAB_PATH,
                                        embed_size=EMBED_SIZE,
                                        hidden_size=HIDDEN_SIZE,
                                        num_layers=NUM_LAYERS):

    # Image preprocessing
    transform = transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.485, 0.456, 0.406),
                                                 (0.229, 0.224, 0.225))])

    # Load vocabulary wrapper
    with open(vocab_path, 'rb') as f:
        vocab = pickle.load(f)

    # Build models

    # eval mode (batchnorm uses moving mean/variance)
    encoder = EncoderCNN(embed_size).eval()
    decoder = DecoderRNN(embed_size, hidden_size, len(vocab), num_layers)
```

```
encoder = encoder.to(device)

decoder = decoder.to(device)


# Load the trained model parameters

encoder.load_state_dict(torch.load(encoder_path))

decoder.load_state_dict(torch.load(decoder_path))


# Prepare an image

image = load_image(image_path, transform)

image_tensor = image.to(device)


# Generate a caption from the image

feature = encoder(image_tensor)

sampled_ids = decoder.sample(feature)


# (1, max_seq_length) -> (max_seq_length)

sampled_ids = sampled_ids[0].cpu().numpy()


# Convert word_ids to words

sampled_caption = []

for word_id in sampled_ids:

        word = vocab.idx2word[word_id]

        sampled_caption.append(word)

        if word == '<end>':

                break

sentence = ' '.join(sampled_caption)[8:-5].title()
```

```
    # Print out the image and the generated caption
    image = Image.open(image_path)
    return sentence, image
```

To predict the labels use following code :

```
plt.figure(figsize=(24,24))
predicted_label, image = PretrainedResNet(image_path='IMAGE_PATH')
plt.imshow(image)
print(predicted_label)
```

Test Image: Thor- Mark I

```
plt.figure(figsize=(17,19))
predicted_label, img = PretrainedResNet(image_path='./image/AVENGERENDGAME1.png')
plt.imshow(img)
print(predicted_label)
```

Test Image: Tony- Mark II

```
plt.figure(figsize=(22,22))
predicted_label, img = PretrainedResNet(image_path='./image/AVENGERENDGAME2.png')
plt.imshow(img)
print(predicted_label)
```

Test Image: Hulk- Mark III

```
plt.figure(figsize=(42,49))
predicted_label, img = PretrainedResNet(image_path='./image/AVENGERENDGAME3.png')
plt.imshow(img)
print(predicted_label)
```

5. Create a Neural network using Python (you can use NumPy to implement this)

Step 1: Import the required libraries

```
# linear algebra
import numpy as np
# data processing, CSV file I / O (e.g. pd.read_csv)
import pandas as pd
import os
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.models import Model
from keras.layers import Flatten, Dense, LSTM, Dropout, Embedding, Activation
from keras.layers import concatenate, BatchNormalization, Input
from keras.layers.merge import add
from keras.utils import to_categorical, plot_model
from keras.applications.inception_v3 import InceptionV3, preprocess_input
import matplotlib.pyplot as plt # for plotting data
import cv2
```

Step 2: Load the descriptions

```
def load_description(text):
    mapping = dict()
    for line in text.split("\n"):
            token = line.split("\t")
            if len(line) < 2: # remove short descriptions
                    continue
            img_id = token[0].split('.')[0] # name of the image
            img_des = token[1]                          # description of the image
            if img_id not in mapping:
                    mapping[img_id] = list()
            mapping[img_id].append(img_des)
```

```
    return mapping


token_path = '/kaggle / input / flickr8k / flickr_data / Flickr_Data / Flickr_TextData / Flickr8k.token.txt'

text = open(token_path, 'r', encoding = 'utf-8').read()

descriptions = load_description(text)

print(descriptions['1000268201_693b08cb0e'])
```

Step 3: Cleaning the text

```
def clean_description(desc):
    for key, des_list in desc.items():
        for i in range(len(des_list)):
            caption = des_list[i]
            caption = [ch for ch in caption if ch not in string.punctuation]
            caption = ''.join(caption)
            caption = caption.split(' ')
            caption = [word.lower() for word in caption if len(word)>1 and word.isalpha()]
            caption = ' '.join(caption)
            des_list[i] = caption


clean_description(descriptions)

descriptions['1000268201_693b08cb0e']
```

Step 4: Generate the Vocabulary

```
def to_vocab(desc):
    words = set()
    for key in desc.keys():
        for line in desc[key]:
            words.update(line.split())
    return words
vocab = to_vocab(descriptions)
```

Step 5: Load the images

```
import glob
images = '/kaggle / input / flickr8k / flickr_data / Flickr_Data / Images/'
# Create a list of all image names in the directory
img = glob.glob(images + '*.jpg')

train_path = '/kaggle / input / flickr8k / flickr_data / Flickr_Data / Flickr_TextData / Flickr_8k.trainImages.txt'
train_images = open(train_path, 'r', encoding = 'utf-8').read().split("\n")
train_img = [] # list of all images in training set
for im in img:
    if(im[len(images):] in train_images):
        train_img.append(im)

# load descriptions of training set in a dictionary. Name of the image will act as ey
def load_clean_descriptions(des, dataset):
    dataset_des = dict()
    for key, des_list in des.items():
        if key+'.jpg' in dataset:
            if key not in dataset_des:
                dataset_des[key] = list()
            for line in des_list:
                desc = 'startseq ' + line + ' endseq'
                dataset_des[key].append(desc)
    return dataset_des

train_descriptions = load_clean_descriptions(descriptions, train_images)
print(train_descriptions['1000268201_693b08cb0e'])
```

Step 6: Extract the feature vector from all images

```
from keras.preprocessing.image import load_img, img_to_array
def preprocess_img(img_path):
    # inception v3 excepts img in 299 * 299 * 3
```

```
    img = load_img(img_path, target_size = (299, 299))
    x = img_to_array(img)
    # Add one more dimension
    x = np.expand_dims(x, axis = 0)
    x = preprocess_input(x)
    return x

def encode(image):
    image = preprocess_img(image)
    vec = model.predict(image)
    vec = np.reshape(vec, (vec.shape[1]))
    return vec

base_model = InceptionV3(weights = 'imagenet')
model = Model(base_model.input, base_model.layers[-2].output)
# run the encode function on all train images and store the feature vectors in a list
encoding_train = {}
for img in train_img:
    encoding_train[img[len(images):]] = encode(img)
```

Step 7: Tokenizing the vocabulary

```
# list of all training captions
all_train_captions = []
for key, val in train_descriptions.items():
    for caption in val:
            all_train_captions.append(caption)

# consider only words which occur atleast 10 times
vocabulary = vocab
threshold = 10 # you can change this value according to your need
word_counts = {}
for cap in all_train_captions:
```

```
    for word in cap.split(' '):
            word_counts[word] = word_counts.get(word, 0) + 1

vocab = [word for word in word_counts if word_counts[word] >= threshold]

# word mapping to integers
ixtoword = {}
wordtoix = {}

ix = 1
for word in vocab:
    wordtoix[word] = ix
    ixtoword[ix] = word
    ix += 1

# find the maximum length of a description in a dataset
max_length = max(len(des.split()) for des in all_train_captions)
max_length
```

Step 8: Glove vector embeddings

```
X1, X2, y = list(), list(), list()
for key, des_list in train_descriptions.items():
    pic = train_features[key + '.jpg']
    for cap in des_list:
            seq = [wordtoix[word] for word in cap.split(' ') if word in wordtoix]
            for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen = max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes = vocab_size)[0]
                    # store
                    X1.append(pic)
                    X2.append(in_seq)
```

```
                    y.append(out_seq)

X2 = np.array(X2)
X1 = np.array(X1)
y = np.array(y)

# load glove vectors for embedding layer
embeddings_index = {}
golve_path ='/kaggle / input / glove-global-vectors-for-word-representation / glove.6B.200d.txt'
glove = open(golve_path, 'r', encoding = 'utf-8').read()
for line in glove.split("\n"):
    values = line.split(" ")
    word = values[0]
    indices = np.asarray(values[1: ], dtype = 'float32')
    embeddings_index[word] = indices

emb_dim = 200
emb_matrix = np.zeros((vocab_size, emb_dim))
for word, i in wordtoix.items():
    emb_vec = embeddings_index.get(word)
    if emb_vec is not None:
        emb_matrix[i] = emb_vec
emb_matrix.shape
```

Step 9: Define the model

```
# define the model
ip1 = Input(shape = (2048, ))
fe1 = Dropout(0.2)(ip1)
fe2 = Dense(256, activation = 'relu')(fe1)
ip2 = Input(shape = (max_length, ))
se1 = Embedding(vocab_size, emb_dim, mask_zero = True)(ip2)
```

```
se2 = Dropout(0.2)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation = 'relu')(decoder1)
outputs = Dense(vocab_size, activation = 'softmax')(decoder2)
model = Model(inputs = [ip1, ip2], outputs = outputs)
```

Step 10: Training the model

```
model.layers[2].set_weights([emb_matrix])
model.layers[2].trainable = False
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')
model.fit([X1, X2], y, epochs = 50, batch_size = 256)
# you can increase the number of epochs for better results
```

Step 11: Predicting the output

```
def greedy_search(pic):
    start = 'startseq'
    for i in range(max_length):
            seq = [wordtoix[word] for word in start.split() if word in wordtoix]
            seq = pad_sequences([seq], maxlen = max_length)
            yhat = model.predict([pic, seq])
            yhat = np.argmax(yhat)
            word = ixtoword[yhat]
            start += ' ' + word
            if word == 'endseq':
                    break
    final = start.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final
```

6. Implement Word Embedding using Word2Vec

Step 1 : By scanning our whole text document and appending the data we create the initial input which we can then transform into a matrix form.

```
import re

def clean_text(
    string: str,
    punctuations=r'''!()-[]{};:'"\,<>./?@#$%^&*_~''',
    stop_words=['the', 'a', 'and', 'is', 'be', 'will']) -> str:
  """
  A method to clean text
  """
  # Cleaning the urls
  string = re.sub(r'https?://\S+|www\.\S+', '', string)

  # Cleaning the html elements
  string = re.sub(r'<.*?>', '', string)

  # Removing the punctuations
  for x in string.lower():
    if x in punctuations:
      string = string.replace(x, "")

  # Converting the text to lower
  string = string.lower()

  # Removing stop words
  string = ' '.join([word for word in string.split() if word not in stop_words])

  # Cleaning the whitespaces
  string = re.sub(r'\s+', ' ', string).strip()

  return string
```

Step 2: The full pipeline to create the (X, Y) word pairs given a list of strings texts:

```
# Defining the window for context
window = 2

# Creating a placeholder for the scanning of the word list
word_lists = []
all_text = []

for text in texts:

  # Cleaning the text
  text = text_preprocessing(text)
```

```
    # Appending to the all text list
    all_text += text

    # Creating a context dictionary
    for i, word in enumerate(text):
        for w in range(window):
            # Getting the context that is ahead by *window* words
            if i + 1 + w < len(text):
                word_lists.append([word] + [text[(i + 1 + w)]])
            # Getting the context that is behind by *window* words
            if i - w - 1 >= 0:
                word_lists.append([word] + [text[(i - w - 1)]])
```

Step 3 : Creation of unique word dictionary

```
def create_unique_word_dict(text:list) -> dict:
    """
    A method that creates a dictionary where the keys are unique words
    and key values are indices
    """
    # Getting all the unique words from our text and sorting them alphabetically
    words = list(set(text))
    words.sort()

    # Creating the dictionary for the unique words
    unique_word_dict = {}
    for i, word in enumerate(words):
        unique_word_dict.update({
            word: i
        })

    return unique_word_dict
```

Step 4: Creating the X and Y matrices

```
from scipy import sparse
import numpy as np

# Defining the number of features (unique words)
n_words = len(unique_word_dict)

# Getting all the unique words
words = list(unique_word_dict.keys())

# Creating the X and Y matrices using one hot encoding
X = []
Y = []

for i, word_list in tqdm(enumerate(word_lists)):
    # Getting the indices
    main_word_index = unique_word_dict.get(word_list[0])
```

```
    context_word_index = unique_word_dict.get(word_list[1])

    # Creating the placeholders
    X_row = np.zeros(n_words)
    Y_row = np.zeros(n_words)

    # One hot encoding the main word
    X_row[main_word_index] = 1

    # One hot encoding the Y matrix words
    Y_row[context_word_index] = 1

    # Appending to the main matrices
    X.append(X_row)
    Y.append(Y_row)

# Converting the matrices into an array
X = np.asarray(X)
Y = np.asarray(Y)
```

Step 5: Training and obtaining weights

```
# Deep learning:
from keras.models import Input, Model
from keras.layers import Dense

# Defining the size of the embedding
embed_size = 2

# Defining the neural network
inp = Input(shape=(X.shape[1],))
x = Dense(units=embed_size, activation='linear')(inp)
x = Dense(units=Y.shape[1], activation='softmax')(x)
model = Model(inputs=inp, outputs=x)
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')

# Optimizing the network weights
model.fit(
    x=X,
    y=Y,
    batch_size=256,
    epochs=1000
    )

# Obtaining the weights from the neural network.
# These are the so called word embeddings

# The input layer
```

```
weights = model.get_weights()[0]

# Creating a dictionary to store the embeddings in. The key is a unique word and
# the value is the numeric vector
embedding_dict = {}
for word in words:
   embedding_dict.update({
      word: weights[unique_word_dict.get(word)]
      })
```

Step 6 : obtain the weights and plot the results:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))

for word in list(unique_word_dict.keys()):

  coord = embedding_dict.get(word)

  plt.scatter(coord[0], coord[1])

  plt.annotate(word, (coord[0], coord[1]))
```

7. Collocations are two or more words that tend to appear frequently together, for example – United States. Implement this using Python

Link to DATA – Monty Python and the Holy Grail script

Code #1 : Loading Libraries

```
from nltk.corpus import webtext

# use to find bigrams, which are pairs of words

from nltk.collocations import BigramCollocationFinder

from nltk.metrics import BigramAssocMeasures
```

Code #2 : Let's find the collocations

```
# Loading the data

words = [w.lower() for w in webtext.words( C:\\Geeksforgeeks\\python_and_grail.txt')]

biagram_collocation = BigramCollocationFinder.from_words(words)

biagram_collocation.nbest(BigramAssocMeasures.likelihood_ratio, 15)
```

Code #3 :

```
from nltk.corpus import stopwords

stopset = set(stopwords.words('english'))

filter_stops = lambda w: len(w) < 3 or w in stopset

biagram_collocation.apply_word_filter(filter_stops)

biagram_collocation.nbest(BigramAssocMeasures.likelihood_ratio, 15)
```

Code #4 : **Working on triplets instead of pairs.**

```
# Loading Libraries
from nltk.collocations import TrigramCollocationFinder
from nltk.metrics import TrigramAssocMeasures
# Loading data - text file
words = [w.lower() for w in webtext.words( C:\Geeksforgeeks\\python_and_grail.txt')]
trigram_collocation = TrigramCollocationFinder.from_words(words)
trigram_collocation.apply_word_filter(filter_stops)
trigram_collocation.apply_freq_filter(3)
trigram_collocation.nbest(TrigramAssocMeasures.likelihood_ratio, 15)
```

8. **WordNet is the lexical database i.e. dictionary for the English language, specifically designed for natural language processing. Synset is a special kind of a simple interface that is present in NLTK to look up words in WordNet. Synset instances are the groupings of synonymous words that express the same concept Show working of these using Python**

Code #1 : Understanding Synset

```
from nltk.corpus import wordnet

syn = wordnet.synsets('hello')[0]

print ("Synset name : ", syn.name())

# Defining the word

print ("\nSynset meaning : ", syn.definition())

# list of phrases that use the word in context

print ("\nSynset example : ", syn.examples())
```

Code #2 : Understanding Hypernerms and Hyponyms from nltk.corpus import wordnet

```
syn = wordnet.synsets('hello')[0]

print ("Synset name : ", syn.name())

print ("\nSynset abstract term : ", syn.hypernyms())

print ("\nSynset specific term : ",

        syn.hypernyms()[0].hyponyms())

syn.root_hypernyms()

print ("\nSynset root hypernerm : ", syn.root_hypernyms())
```

Code #3 : Part of Speech (POS) in Synset.

```
syn = wordnet.synsets('hello')[0]

print ("Syn tag : ", syn.pos())

syn = wordnet.synsets('doing')[0]
```

```
print ("Syn tag : ", syn.pos())

syn = wordnet.synsets('beautiful')[0]

print ("Syn tag : ", syn.pos())

syn = wordnet.synsets('quickly')[0]

print ("Syn tag : ", syn.pos())
```

9. Implement Naïve Baye's Classifier using python.

Download dataset here

```
# Importing library
import math
import random
import csv


# the categorical class names are changed to numberic data
# eg: yes and no encoded to 1 and 0
def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
        if mydata[i][-1] not in classes:
            classes.append(mydata[i][-1])
    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata


# Splitting the data
def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    # initially testset will have all the dataset
    test = list(mydata)
    while len(train) < train_num:
        # index generated randomly from range 0
```

```
            # to length of testset
            index = random.randrange(len(test))
            # from testset, pop data rows and put it in train
            train.append(test.pop(index))
    return train, test


# Group the data rows under each class yes or
# no in dictionary eg: dict[yes] and dict[no]
def groupUnderClass(mydata):
    dict = {}
    for i in range(len(mydata)):
            if (mydata[i][-1] not in dict):
                    dict[mydata[i][-1]] = []
            dict[mydata[i][-1]].append(mydata[i])
    return dict


# Calculating Mean
def mean(numbers):
    return sum(numbers) / float(len(numbers))

# Calculating Standard Deviation
def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)

def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    # eg: list = [ [a, b, c], [m, n, o], [x, y, z]]
```

```
    # here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3
    # delete summaries of last class
    del info[-1]
    return info

# find Mean and Standard Deviation under each class
def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info


# Calculate Gaussian Probability Density Function
def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo


# Calculate Class Probabilities
def calculateClassProbabilities(info, test):
    probabilities = {}
    for classValue, classSummaries in info.items():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities
```

```
# Make prediction - highest probability is the prediction
def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel


# returns predictions for a set of examples
def getPredictions(info, test):
    predictions = []
    for i in range(len(test)):
        result = predict(info, test[i])
        predictions.append(result)
    return predictions

# Accuracy score
def accuracy_rate(test, predictions):
    correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test))) * 100.0


# driver code
```

```
# add the data path in your system
filename = r'E:\user\MACHINE LEARNING\machine learning algos\Naive bayes\filedata.csv'



# load the file and store it in mydata list
mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
    mydata[i] = [float(x) for x in mydata[i]]



# split ratio = 0.7
# 70% of data is training data and 30% is test data used for testing
ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))

# prepare model
info = MeanAndStdDevForClass(train_data)

# test model
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

10. Twitter Sentiment Analysis using Python. Fetch tweets from twitter using Python and implement it.

Installation

- **Tweepy:** tweepy is the python client for the official Twitter API.
  Install it using following pip command:
  pip install tweepy
- **TextBlob:** textblob is the python library for processing textual data.
  Install it using following pip command:
  pip install textblob

  Also, we need to install some NLTK corpora using following command:

  python -m textblob.download_corpora

Authentication:

In order to fetch tweets through Twitter API, one needs to register an App through their twitter account. Follow these steps for the same:

- Open this link and click the button: 'Create New App'
- Fill the application details. You can leave the callback url field empty.
- Once the app is created, you will be redirected to the app page.
- Open the 'Keys and Access Tokens' tab.
- Copy 'Consumer Key', 'Consumer Secret', 'Access token' and 'Access Token Secret'.

```
import re

import tweepy

from tweepy import OAuthHandler

from textblob import TextBlob


class TwitterClient(object):
        '''
        Generic Twitter Class for sentiment analysis.
        '''
        def __init__(self):
                '''
                Class constructor or initialization method.
                '''
                # keys and tokens from the Twitter Dev Console
                consumer_key = 'XXXXXXXXXXXXXXXXXXXXXXXXX'
                consumer_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
                access_token = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
```

```
                access_token_secret = 'XXXXXXXXXXXXXXXXXXXXXXXXX'

                # attempt authentication
                try:
                        # create OAuthHandler object
                        self.auth = OAuthHandler(consumer_key, consumer_secret)
                        # set access token and secret
                        self.auth.set_access_token(access_token, access_token_secret)
                        # create tweepy API object to fetch tweets
                        self.api = tweepy.API(self.auth)
                except:
                        print("Error: Authentication Failed")

        def clean_tweet(self, tweet):
                '''
                Utility function to clean tweet text by removing links, special characters
                using simple regex statements.
                '''
                return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])
                                                                |(\w+:\/\/\S+)",       "       ",
tweet).split())

        def get_tweet_sentiment(self, tweet):
                '''
                Utility function to classify sentiment of passed tweet
                using textblob's sentiment method
                '''
                # create TextBlob object of passed tweet text
                analysis = TextBlob(self.clean_tweet(tweet))
                # set sentiment
                if analysis.sentiment.polarity > 0:
```

```
                return 'positive'
        elif analysis.sentiment.polarity == 0:
                return 'neutral'
        else:
                return 'negative'

def get_tweets(self, query, count = 10):
        '''
        Main function to fetch tweets and parse them.
        '''
        # empty list to store parsed tweets
        tweets = []

        try:
                # call twitter api to fetch tweets
                fetched_tweets = self.api.search(q = query, count = count)

                # parsing tweets one by one
                for tweet in fetched_tweets:
                        # empty dictionary to store required params of a tweet
                        parsed_tweet = {}

                        # saving text of tweet
                        parsed_tweet['text'] = tweet.text
                        # saving sentiment of tweet
                        parsed_tweet['sentiment'] = self.get_tweet_sentiment(tweet.text)

                        # appending parsed tweet to tweets list
                        if tweet.retweet_count > 0:
                                # if tweet has retweets, ensure that it is appended only once
                                if parsed_tweet not in tweets:
```

```
                                        tweets.append(parsed_tweet)
                                else:
                                        tweets.append(parsed_tweet)

                        # return parsed tweets
                        return tweets

                except tweepy.TweepError as e:
                        # print error (if any)
                        print("Error : " + str(e))

def main():
        # creating object of TwitterClient Class
        api = TwitterClient()
        # calling function to get tweets
        tweets = api.get_tweets(query = 'Donald Trump', count = 200)

        # picking positive tweets from tweets
        ptweets = [tweet for tweet in tweets if tweet['sentiment'] == 'positive']
        # percentage of positive tweets
        print("Positive tweets percentage: {} %".format(100*len(ptweets)/len(tweets)))
        # picking negative tweets from tweets
        ntweets = [tweet for tweet in tweets if tweet['sentiment'] == 'negative']
        # percentage of negative tweets
        print("Negative tweets percentage: {} %".format(100*len(ntweets)/len(tweets)))
        # percentage of neutral tweets
        print("Neutral tweets percentage: {} % \
                ".format(100*(len(tweets) -(len( ntweets )+len( ptweets)))/len(tweets)))

        # printing first 5 positive tweets
        print("\n\nPositive tweets:")
```

```
        for tweet in ptweets[:10]:
                print(tweet['text'])

        # printing first 5 negative tweets
        print("\n\nNegative tweets:")
        for tweet in ntweets[:10]:
                print(tweet['text'])

if __name__ == "__main__":
        # calling main function
        main()
```