Ex No: 1	Build Logistic Regression Classifier to recognise cats

AIM:

To build a logistic regression classifier capable of recognizing images containing cats.

PROCEDURE:

1. Import the required libraries and packages.
2. Create helper functions to fetch the dataset from the specified Amazon S3 URL.
3. Download the dataset in zip format.
4. Extract and load the dataset as a list of dog images and cat images separately.
5. Use 7000 dog images and 7000 cat images for training.
6. Combine the dog images set and cat images set and shuffle them randomly.
7. Create an array named `train_data` that will contain all zeros.
8. Iterate through combined train list and crop each image from center in 100x100 size, then convert PIL object into NumPy array and store it into train_data.
9. Scale array values between 0 and 1 by dividing all the elements by 255 so that algorithm can converge nicely.
10. Label the data: `cat` as 0 and `dog` as 1.
11. Build the logistic regression model.
12. Train the model with the training data.
13. Obtain the training data accuracy.
14. Test the model with the test data and obtain the test data accuracy.
15. Predict own custom image using the helper functions and model.

CODE:

```
# importing useful libraries

import requests

import zipfile

import os

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix

import seaborn as sns

np.random.seed(42)



#Helper functions

def download(url, file_name):
```

```
  """Downloads the dataset for project input: url (string): url for dataset file_name (string): file name in which downloaded dataset will be stored"""
  response = requests.get(url, stream=True)
  if response.status_code == 200:
      with open(file_name, 'wb') as f:
          f.write(response.raw.read())
def extract_zip(s_path, d_path):
  """Extract (unzip) the compressed dataset for working input:
s_path (string): path of zipped dataset
d_path (string): path to store the unzipped dataset"""
  with zipfile.ZipFile(s_path, 'r') as zip_ref:
      zip_ref.extractall(d_path)
def center_crop(image_path, size):
  """crop the image from center of the given size input: image_path (string): Path of image size (int): size to which the image being cropped from center"""
  img = Image.open(image_path)
  img = img.resize((size+1,size+1))
  x_center = img.width/2
  y_center = img.height/2
  size = size/2
  cr = img.crop((x_center-size, y_center-size, x_center+size, y_center+size))
 return cr

# downloading and extracting dataset
download("https://s3.amazonaws.com/content.udacity-data.com/nd089/Cat_Dog_data.zip", "images.zip")
extract_zip("images.zip", "images")
Dislpaying an image just for verification of dataset
img = Image.open("./images/Cat_Dog_data/train/cat/cat.0.jpg")
img.format, type(img)
plt.imshow(img)

# Grabbing all the names of cats and dogs images
cat_list = os.listdir("./images/Cat_Dog_data/train/cat")
dog_list = os.listdir("./images/Cat_Dog_data/train/dog")
```

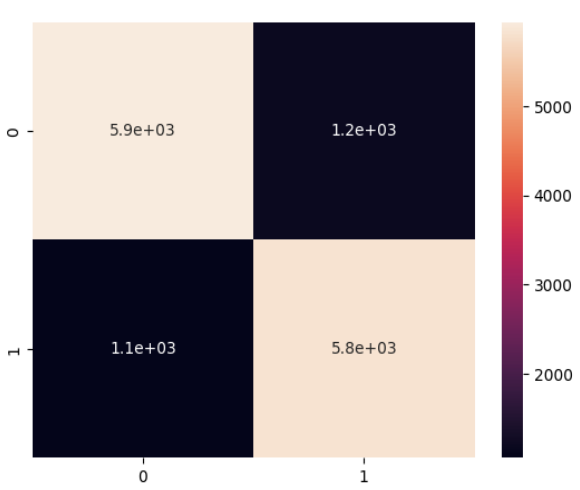
```
cat_list = cat_list[:7000]
dog_list = dog_list[:7000]
len(cat_list), len(dog_list)
train_list = []
train_list.extend(cat_list)
train_list.extend(dog_list)
np.random.shuffle(train_list)
## shuffled version of `train_list`
train_list[:10]

# Preparing Training data
train_data = np.zeros((14000,100*100*3))
for i, image_name in enumerate(train_list):
  if image_name.split(".")[0] == "dog":
    path =  "./images/Cat_Dog_data/train/dog"
  else:
    path =  "./images/Cat_Dog_data/train/cat"
  image_path = f'{path}/{image_name}'
  crp_img = center_crop(image_path,100)
  crp_arr = np.array(crp_img).reshape(-1)
  train_data[i] = crp_arr
train_data[0]
train_data = train_data/255
train_data[0]

# Preparing label data
print("printing the name of some image")
print("-> ",train_list[0])
print("Splitting the image from all . characters into a list")
print("-> ",train_list[0].split("."))
print("selecting the 0th element of splitted list")
print("-> ",train_list[0].split(".")[0])
# cat: 0
# dog: 1
```

```
train_labels = np.array([0 if name.split(".")[0]=="cat" else 1 for name in train_list])
train_labels.shape
model = LogisticRegression(max_iter=300, n_jobs=-1)
model.fit(train_data, train_labels)
model.score(train_data, train_labels)

# Predicting an image to see the result
test_img = "./images/Cat_Dog_data/test/cat/cat.10038.jpg"
img = Image.open(test_img)
plt.imshow(img)
im = center_crop(test_img,100) # cropping image
X = np.array(im).reshape(-1) # flattening the image to pass in model for prediction
X = X/255 # scale the pixels in 0-1 range
model.predict([X])

# Analyzing results on training data using Confusion matrix
train_pred = model.predict(train_data)
cm = confusion_matrix(train_pred, train_labels)
cm
sns.heatmap(cm, annot=True)

# Preparing test data to check performance on unseen samples
test_data_cat = os.listdir("./images/Cat_Dog_data/test/cat")
test_data_dog = os.listdir("./images/Cat_Dog_data/test/dog")
test_data_cat = test_data_cat[:1250]
test_data_dog = test_data_dog[:1250]
test_list = []
test_list.extend(test_data_cat)
test_list.extend(test_data_dog)
len(test_list)
test_data = np.zeros((2500,100*100*3))
for i, image_name in enumerate(test_list):
  if image_name.split(".")[0] == "dog":
    path =  "./images/Cat_Dog_data/test/dog"
```

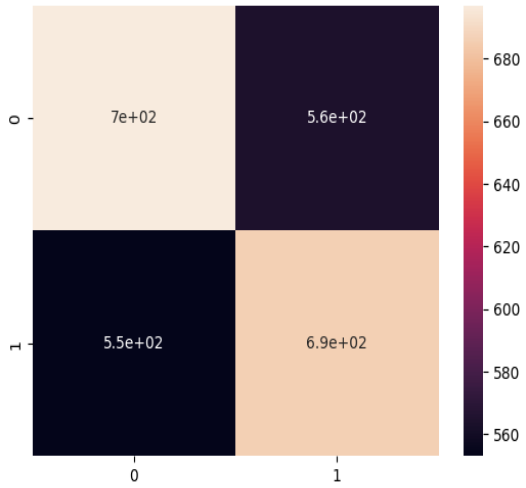
```
  else:
    path =  "./images/Cat_Dog_data/test/cat"
  image_path = f'{path}/{image_name}'
  crp_img = center_crop(image_path,100)
  crp_arr = np.array(crp_img).reshape(-1)
  test_data[i] = crp_arr
test_data = test_data/255
test_labels = np.array([0 if name.split(".")[0]=="cat" else 1 for name in test_list])

# Analyzing results on test data using Confusion matrix
pred = model.predict(test_data)
cm = confusion_matrix(pred, test_labels)
print(cm)
sns.heatmap(cm, annot=True)
test_acc = model.score(test_data, test_labels)
print("Accuracy on test set: ", test_acc)

# Predicting own custom image
def show_image(img_path):
    img = Image.open(img_path)
    plt.imshow(img)
def predict_custom_image(model, img_path):
    crp_img = center_crop(img_path,100)
    crp_arr = np.array(crp_img).reshape(1,-1)
    pred = model.predict(crp_arr)
    if pred == 0:
        return "Cat"
    return "Dog"
test_img_path = "test.jpg" # provide path to your custom image (make sure it's either jpg or jpeg)
show_image(test_img_path)
predict_custom_image(model, test_img_path)
```

OUTPUT:

Confusion matrix heatmap



Training



Testing

Predicted: 'Cat'

RESULTS:

Accuracy for	Score
Training	0.8396428571428571
Testing	0.5532

INFERENCES:

1. The accuracy is low for logistic regression-based cat/dog classification because:
   - Captures only linear combinations of features.
   - Doesn't capture complex, non-linear relationships between features.
2. Feature engineering is required.
3. Needs High dimensional algorithms like CNNs.

CONCLUSION:

A logistic regression classifier capable of recognizing images containing cats has been successfully built and implemented.