ENIN 880CA Assignment #1

Elyar Zavvari

November 14, 2021

Students should read chapters 1 to 7 (all inclusive) from the PDF file sent to the students' list via email. Students should watch videos on chapters 1 to 7, shared via Dropbox. Students should learn about the code explained in chapter 7 inside out, and then copy and run the code (i.e., image classification using K-NN method).

Students should follow the following structure for folders and files they create:

- Root folder: It should include the file: "knn.py" and the folder: "pyimagesearch".
- Folder "pyimagesearch should include the file: "-init-.py" and two folders: "datasets" and "preprocessing".
- The folder "datasets" should include files: "-init-.py" and "simpledatasetloader.py".
- The folder "preprocessing" should include two files: "-init-.py" and "simplepreprocessor.py"

Using an image dataset that includes images of "cats", "dogs", and "pandas", insert an unknown image (e.g. an image of a dog for instance) into the image dataset and then make a classification. Repeat this for random pictures of cats and pandas. Which class will give you the highest rate of success, and why?

NOTE: the purpose for this exercise is to learn the basics of the K-NN for classification. K-NN works fast, but it does not actually "learn" anything (i.e., an open-loop classification algorithm with no back propagation of parameters and/or feedback). However, it is the simplest "classification" algorithm one can think of and start with. People usually use K-NN as a baseline.

Solution:*As you can see in the repository a main folder created named "pyimagesearch". Inside this folder yo'll find the python codes and a folder named "pics". Inside the "pics" folder we have our dataset which are "cat". "dog", and "panda". In each of them there are 100 pics of related dataset tag. A portion of these dataset are being used to train the model and rest of them for testing the model.*
One of the python codes is "simplepreprocessor.py". This code is used to make the dimension of all pictures same. The model wants to compare all the pictures, as a result their dimensions should be harmonized. The "simplepreprocessor.py" is as follows:

1

```python
import cv2

class SimplePreprocessor:
    def __init__(self, width, height, inter=cv2.INTER_AREA):
        # store the target image width, height, and interpolation #
        # method used when resizing #
        self.width = width
        self.height = height
        self.inter = inter

    def preprocess(self, image):
        # resize the image to a fixed size, ignoring the aspect #
        # ratio #
        return cv2.resize(image, (self.width, self.height),
            interpolation=self.inter)
```

The second python code is "simpledatasetloader.py". This script loads the pictures and at the same time excludes the label for the loaded picures. "simpledatasetloader.py" script is as follows:

1

```
# import the necessary packages #
import numpy as np
import cv2
import os

class SimpleDatasetLoader:
    def __init__(self, preprocessors=None):
        # store the image preprocessor #
        self.preprocessors = preprocessors

        # if the preprocessors are None, initialize them as an #
        # empty list #
        if self.preprocessors is None:
            self.preprocessors = []

    def load(self, imagePaths, verbose=-1):
        # initialize the list of features and labels #
        data = []
        labels = []

        # loop over the input images #
        for (i, imagePath) in enumerate(imagePaths):
            # load the image and extract the class label assuming #
            # that our path has the following format: #
            # /path/to/dataset/{class}/{image}.jpg #
            image = cv2.imread(imagePath)
            label = imagePath.split(os.path.sep)[-2]

            # check to see if our preprocessors are not None #
            if self.preprocessors is not None:
                # loop over the preprocessors and apply each to #
                # the image
                for p in self.preprocessors:
                    image = p.preprocess(image)
            # treat our processed image as a "feature vector" #
            # by updating the data list followed by the labels #
            data.append(image)
            labels.append(label)

            # show an update every `verbose` images #
            if verbose > 0 and i > 0 and (i + 1) % verbose == 0:
                print("[INFO] processed {}/{}".format(i + 1,
                    len(imagePaths)))

        # return a tuple of the data and labels #
        return (np.array(data), np.array(labels))
```

Finally the "knn.py" scripts calls "simpledatasetloader.py" and "simplepreprocessor.py" packagesand by importing the required packages applies the KNN classification approach. "kn.py" python codes are as follows:

1

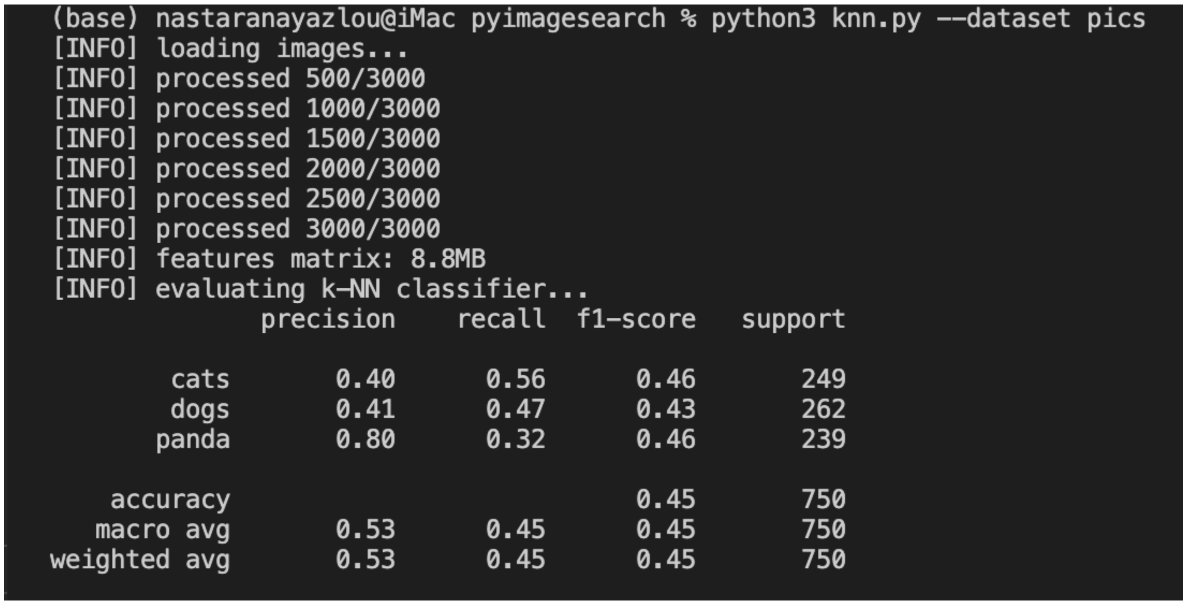
```
# import the necessary packages #
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from simplepreprocessor import SimplePreprocessor
from simpledatasetloader import SimpleDatasetLoader
from imutils import paths
import argparse

# construct the argument parse and parse the arguments #
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
ap.add_argument("-k", "--neighbors", type=int, default=1,
    help="# of nearest neighbors for classification")
ap.add_argument("-j", "--jobs", type=int, default=-1,
    help="# of jobs for k-NN distance (-1 uses all available cores)")
args = vars(ap.parse_args())

# grab the list of images that we'll be describing #
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))

# initialize the image preprocessor, load the dataset from disk, #
# and reshape the data matrix #
sp = SimplePreprocessor(32, 32)
sdl = SimpleDatasetLoader(preprocessors=[sp])
(data, labels) = sdl.load(imagePaths, verbose=500)
data = data.reshape((data.shape[0], 3072))

# show some information on memory consumption of the images #
print("[INFO] features matrix: {:.1f}MB".format(
    data.nbytes / (1024 * 1024.0)))

# encode the labels as integers #
le = LabelEncoder()
labels = le.fit_transform(labels)

# partition the data into training and testing splits using 75% of #
# the data for training and the remaining 25% for testing #
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.25, random_state=42)

# train and evaluate a k-NN classifier on the raw pixel intensities #
print("[INFO] evaluating k-NN classifier...")
```

```
model = KNeighborsClassifier(n_neighbors=args["neighbors"],
    n_jobs=args["jobs"])
model.fit(trainX, trainY)
print(classification_report(testY, model.predict(testX),
    target_names=le.classes_))
```

Results: In order to run the scripts run the following code inside ”pyimagesearch” folder:

python3 knn.py - -dataset pics

the result would be:



```
(base) nastaranayazlou@iMac pyimagesearch % python3 knn.py --dataset pics
[INFO] loading images...
[INFO] processed 500/3000
[INFO] processed 1000/3000
[INFO] processed 1500/3000
[INFO] processed 2000/3000
[INFO] processed 2500/3000
[INFO] processed 3000/3000
[INFO] features matrix: 8.8MB
[INFO] evaluating k-NN classifier...
              precision    recall  f1-score   support

        cats       0.40      0.56      0.46       249
        dogs       0.41      0.47      0.43       262
       panda       0.80      0.32      0.46       239

    accuracy                           0.45       750
   macro avg       0.53      0.45      0.45       750
weighted avg       0.53      0.45      0.45       750
```