The University of Regina
Faculty of Engineering and Applied Science
Applied Machine Learning
10-11 December 2020; SK/Regina, Canada https://www.uregina.ca/engineering/

K-NN Algorithm for Image Classification

(Final Project)

Behnam Moradi (*Author*)
Faculty of Eng. and Applied Science
Industrial Systems Engineering
bmn891@uregina.ca

Abstract

KNN classification algorithm is one of the well known classification methods in data engineering and image processing. This algorithm is basically a start point to use ML algorithms in the world of image processing. In this project, the KNN algorithm has been used to classify 3 kinds of animals namely, cat, dog, and panda. The animal dataset including 3000 images(1000 images per animal) has been used. The KNN algorithm has been written in python3. The results show that the KNN performance in image classification can be improved by increasing the number of nearest neighbors K to a specific number. Having said the performance cannot be improved by increasing K beyond the extracted criteria. Having said that the performance has a direct relation with the data quality.

Keyword; KNN; Machine_learning; image_classification;

I. INTRODUCTION

When we are referring to Data classification, basically, we are referring to a technique for assigning a class to a particular data which belongs to a particular category. This is the most basic principle in data analysis. Extracting a separation Surface/Line/Curve which divides a feature space into multiple regions. This technique can be applied to a wide range of classification problems and data structures including Image classification.

The main focus of this project is using data classification to classify Images. Image Classification is a basic data classification technique that tries to consider an image like a multidimensional array as a whole. Each pixel will be considered as one specific dimension and the pixel brightness will be considered the vector amplitude on this dimension.

The objective is to classify an unknown image by assigning it to a specific category/class/label. Typically, Image Classification refers to images in which only one object appears and is analyzed. Based on this concept, the author will be using an animal dataset in which there exists one specific animal in a single image.

K nearest neighbors algorithm will be used in this project to perform the mentioned image classification task. It is a simple algorithm that imports all the training dataset and classifies unknown images based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique [1].

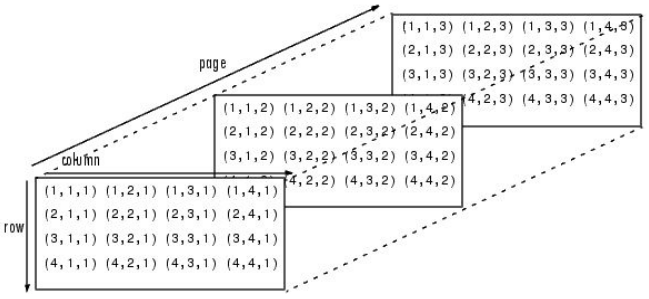
II. METHODS

A. OpenCV and Image Processing

When it comes to image classification, at the very first step, we need to understand what an image is and how we can read it information mathematically for the further proceedings. Technically, an image is simply a 3D Matrix filled by elements starting from 0 to 255. In image processing terminology, each dimension of this 3D matrix represents one single color channel. Each image has 3 color channels: R, G, and B:

- R: Red channel
- G: Green channel
- B: Blue channel

Having said that, each single element represents its respective pixel within the image. The following figure illustrates the overall mathematical idea behind an image;



Each matrix represents one single color channel and each element represents the pixel brightness starting from 0 to 255.

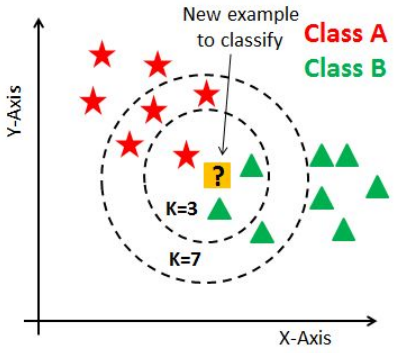
In this project, in order to reduce the number of pixels and import images from the dataset, OpenCV (cv2) library has been used. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

B. Image Classification Techniques

Classification between objects is a complex task and therefore image classification has been an important task within the field of computer vision. Image classification refers to the labelling of images into one of a number of predefined classes. There are potentially n number of classes in which a given image can be classified. Manually checking and classifying images could be a tedious task especially when they are massive in number and therefore it will be very useful if we could automate this entire process using computer vision.
To be clear about the terminology, we refer to object classification as a Supervised classification technique.
[https://medium.com/]
Supervised classification uses classification algorithms and regression techniques to develop predictive models. The algorithms include linear regression, logistic regression, neural networks, decision tree, support vector machine, random forest, naive Bayes, and k-nearest neighbor. As the title of this project suggests, K-Nearest Neighbors (KNN) classification has been used to classify a single object inside a single image by assigning a label to the image.

C. K-Nearest Neighbors

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. KNN, technically, does not learn any parameters. It simply classifies data points based on the points that are most similar to it. It uses test data to make an "educated guess" on what an unclassified point should be classified as. The following figure illustrate the basic idea behind KNN;



As it can be seen from the figure, KNN classifies an unknown datapoint based on its distance from the training data. The only parameter to learn is the K which is the number of nearest data points. One the algorithm finds these data points, it simply goes through a voting process and selects the class which has the higher number of votes.
In order to calculate the distance between data points, there are three kinds of distance function to be used within the algorithm.

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

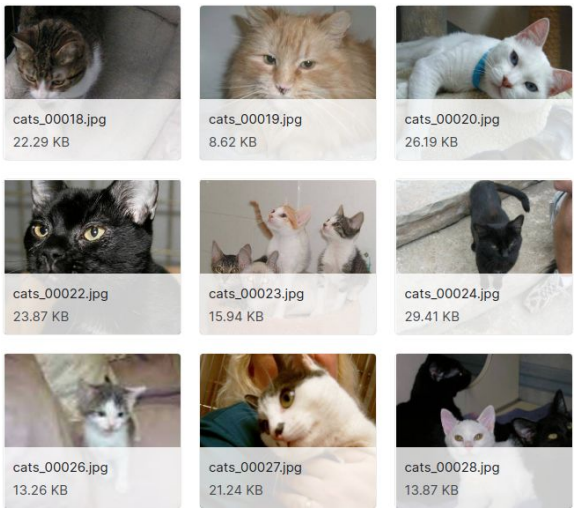
Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

We will be simply using the Euclidean distance function in this project. KNN has 3 main advantages that makes it very useful for simple data classification; It is very easy to use and implement, very quick and computationally low cost, and above all it simply does not make any assumption about the data.

III. DATASET

The Animal dataset is a very widely used machine learning oriented dataset for image classification using both traditional and modern classification techniques. This dataset consists of 3000 images and it covers three classes of animals: Cats, Dogs, and Panda. Each class has 1000 images which makes the dataset suitable for CPU processing. Our regular computers are able to simply load them on the RAM and one does not need to use GPU based computers to work with dataset. One is able to simply download the Animal dataset from https://www.kaggle.com/.

The following figures illustrate some sample cats images of this dataset.



IV. Code Explained

The main focus of this project is to write a python-based algorithm for KNN image classification technique in order to classify 3 major animals: Cats, Dogs, and Panda. The following system configuration has been used in this project:

- Lenovo LEGION Y540
- RAM: 16
- CPU: Core i7 9th Gen
- OS: Ubuntu 18.04
- IDE: VS Code

All the libraries have been installed using "pip3" and the programming language is python3.6. Having said that the author is mainly using Docker and ADE wrapper for his developments. Running the script from Ubuntu Docker Image 18.04 is recommended. Otherwise running the script from the main Ubuntu terminal is quite good and straightforward.

It is important to mention that the following strategies are considered to be the technical scope of this project:

1. Writing the overall structure of the algorithm
2. Manipulating the algorithm with different numbers of nearest neighbors K starting from 0 to 30.
3. Manipulating the percentage of training-test data

The results will be based on the following metrics for every animal separately;

1. Precision
2. Recall
3. F1-Score

$$PRE = \frac{TP}{TP + FP}$$

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

A. Libraries

In this project, the following libraries has been used by the author:

```
import os
import cv2
import numpy as np

import argparse
from imutils import paths
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

cv2: Image processing library; This library will be used to read the images and load the entire animal dataset. Also, reducing the number of pixels associated with single images is another application of this library in this project.
imutils: According to anaconda references, imutils is another image processing library wich is way simpler than OpenCV and can be used to do basic image processing tasks such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much more easier with OpenCV and both Python 2.7 and Python 3.
numpy: This library is the most famous algebraic mathematical library that has been very widely used in python programming. This library is important in particular for working with matrices and to plot diagrams. Numpy also supports multi-dimensional arrays, along with a large collection of high-level mathematical functions to operate on these arrays.
sklearn: Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

In this project, the author relied on functions that are provided by sklearn library for KNN image classification. All the functions tutorials have been extracted from sklearn official page.

B. Loading Dataset

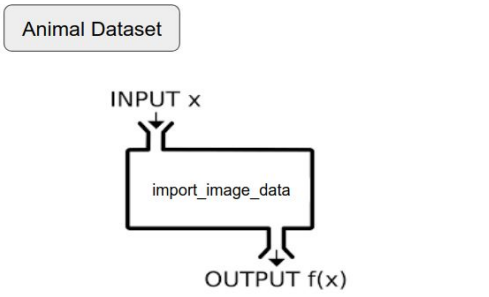
In order to load images from animal dataset, a function has been created and used within the main code structure. As it can be seen from the code block, in order to reduce the computation cost, the images pixels have been reduced to 32x32 matrix structure. Also, the image name has been simply used as the label for the image. The output of this function are the followings:

- Image Data
- Image labels

The following script and figure illustrate how this function works:

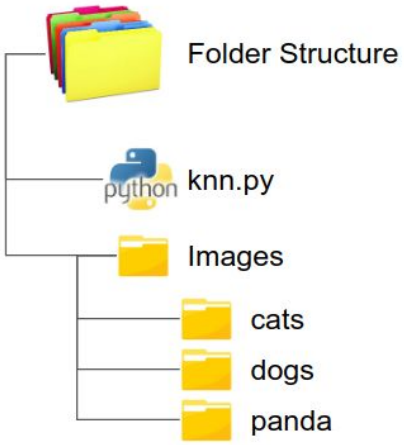
```
def import_images_data(samples_directory):

    image_pixels = []
    image_labels = []
```

```
    for i, image_path in enumerate(samples_directory):
        sample_image = cv2.imread(image_path)
        sample_label = image_path.split(os.path.sep)[-2]

        sample_image = cv2.resize(sample_image, (32, 32),
interpolation=cv2.INTER_AREA)

        image_pixels.append(sample_image)
        image_labels.append(sample_label)

    return (np.array(image_pixels), np.array(image_labels))
```



C. *Preparing the dataset*

To keep everything as simple as possible, all the images are placed inside a folder called “Images”. The following structure illustrate the dataset structure:




```
# Reading images from Image folder located inside the knn.py file
path_to_sample_images = 'Images'
# Directories for every single image
samples_directory = list(paths.list_images(path_to_sample_images))

# Seperating and saving image pixels and image labels
(image_pixels, image_labels) = import_images_data(samples_directory)

# Reshaping image pixels
image_pixels = image_pixels.reshape((image_pixels.shape[0], 3072))

# Every image label will be incoded to a single digit: Cats: 0, Dogs: 1, Panda:
2
encod_image_labels = LabelEncoder()
image_labels = encod_image_labels.fit_transform(image_labels)
```

```
# Number of neighbors to be manipulated by the programmer
number_of_neighbors = 10

# Spliting dataset into 75% training data and 25% test data. It can be changed.
(trainig_image_pixels, test_image_pixels, training_image_labels,
test_image_labels) = train_test_split(image_pixels, image_labels,
test_size=0.25, random_state=42)
```

As it can be seen from the core block, the image matrix has been reshaped to one 1D vector to simplify working with its elements. The image labels are also encoded from their labels to a respective integer digit as follows:

- cats: 0
- dogs: 1
- panda: 2

and finally the data set has been splitted into 75% training data and 25% test data. THis is done by the train_test_split() function. The label and data associated with all the dataset are the inputs and then the label and image data will be splitted into 2 subset of training and test data.

D. *Metrics*

The following variable have been used to store the output of the classification algorithm:

```
# Total Metrics
total_precision_array = []
total_recal_array = []
total_f1_score_array = []

# Precision Metric
cats_precision_array = []
dogs_precision_array = []
pandas_precision_array = []

# Recall Metric
cats_recall_array = []
dogs_recall_array = []
pandas_recall_array = []

# F1-Score Metric
cats_f1_score_array = []
dogs_f1_score_array = []
pandas_f1_score_array = []
```

E. *Creating the Model*

The sklearn library has been used to create the KNN model. The following strategy has been followed to generate results for making comparison:
The KNN classifier has been evaluated using a different number of neighbors in each iteration starting from 1 to 30. As it can be seen from the code block, at the very first step, the knn classifier has been created. Then the training data has been imported to the classifier. Once the training process is done, the prediction stage has begun. At the final stage, the classification report has been created to present the results.

```
# Classification will strt from here
print("Starting Classification...")
for i in range(1,30):
    # Creating KNN classifier
    knn_classifier = KNeighborsClassifier(n_neighbors=i)

    # Model fitting
```

```
    knn_classifier.fit(trainig_image_pixels,
training_image_labels)

    # Making predictions
    prediction = knn_classifier.predict(test_image_pixels)

    # Saving correct predictions
    correct_predictions = np.sum(prediction ==
test_image_labels)

    # FYI: This is how percision in being done. I will use the
library function to do this step
    precision =
float(correct_predictions)/float(test_image_labels.size)

    # Outputing the report
    classifier_report =
classification_report(test_image_labels, prediction,
target_names=encod_image_labels.classes_, output_dict=True)

    # Saving results for precision

cats_precision_array.append(classifier_report["cats"]["precisi
on"])

dogs_precision_array.append(classifier_report["dogs"]["precisi
on"])

pandas_precision_array.append(classifier_report["panda"]["prec
ision"])

    # Saving results for recall

cats_recall_array.append(classifier_report["cats"]["recall"])

dogs_recall_array.append(classifier_report["dogs"]["recall"])

pandas_recall_array.append(classifier_report["panda"]["recall"
])

    # Saving results for f1-score

cats_f1_score_array.append(classifier_report["cats"]["f1-score
"])

dogs_f1_score_array.append(classifier_report["dogs"]["f1-score
"])


pandas_f1_score_array.append(classifier_report["panda"]["f1-sc
ore"])
```

F. *Illustrations*

At the final step of KNN classification, the results have been provided in the following concepts:

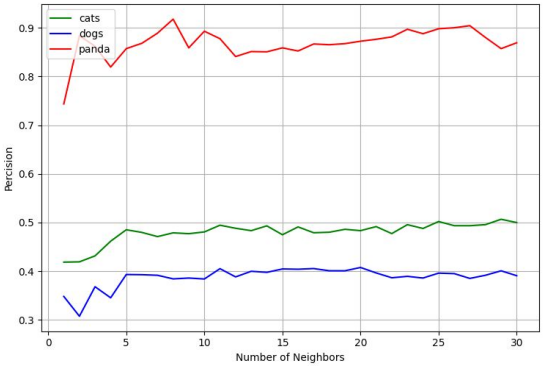
- Precision
- Recall
- F1-Score

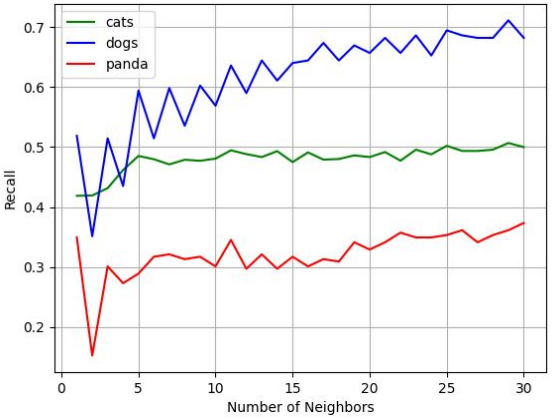
```
# Data visualization - precision
plt.plot(range(1, len(cats_precision_array)+1)
,cats_precision_array, c="green")
plt.plot(range(1, len(dogs_precision_array)+1)
,dogs_precision_array, c="blue")
plt.plot(range(1, len(pandas_precision_array)+1)
,pandas_precision_array, c="red")
plt.xlabel("Number of Neighbors")
plt.ylabel("Percision")
plt.show()

# Data visualization - recall
plt.plot(range(1, len(cats_precision_array)+1)
,cats_precision_array, c="green")
plt.plot(range(1, len(dogs_recall_array)+1)
,dogs_recall_array, c="blue")
plt.plot(range(1, len(pandas_recall_array)+1)
,pandas_recall_array, c="red")
plt.xlabel("Number of Neighbors")
plt.ylabel("Recall")
plt.show()

# Data visualization - f1_score
plt.plot(range(1, len(cats_f1_score_array)+1)
,cats_f1_score_array, c="green")
plt.plot(range(1, len(dogs_f1_score_array)+1)
,dogs_f1_score_array, c="blue")
plt.plot(range(1, len(pandas_f1_score_array)+1)
,pandas_f1_score_array, c="red")
plt.xlabel("Number of Neighbors")
plt.ylabel("F1_score")
plt.show()
```

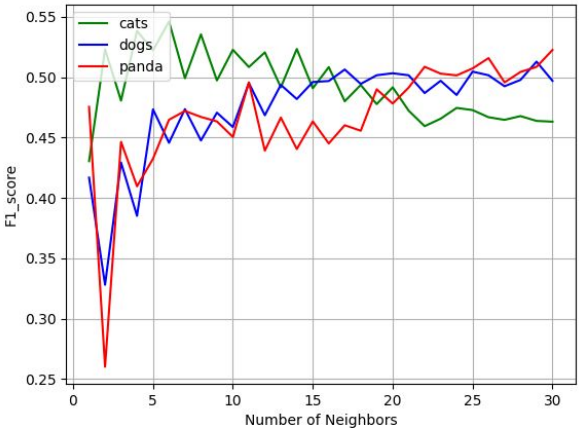
V. Results and Conclusion

In this project, the KNN classification algorithm was successfully used to classify 3 kinds of animals. The algorithm performance can be derived from precision, recall, and f1-score metrics. The following figures are associated with those metrics for each animal. As it can be seen from the precision diagram, this factor can be improved by increasing the number of nearest neighbors until it reaches a specific point. The recall and f1-score are also increasing until they reach a specific point. Based on these results, for every KNN classification project, there is a specific criteria for the number K and the data engineer should try to find this number.

To sum up the project results, it is clear that the KNN method is not a good option for image classification, especially when the image is not very clear and straightforward. For example, the panda images are very simple, clear, and easy to be classified by KNN. As it can be seen from the precision diagram, KNN achieves up to 90% precision which is very accurate. But for other images, dogs and cats, the precision is very low and it is not reliable.







VI. References

[1] https://www.tutorialspoint.com/

[2] https://opencv.org/about/

[3] https://www.kaggle.com/