Image classification

Learning Objectives

At the end of this session you will be able to:

- Familiarized with installation missing packages
- Perform image processing
- Perform feature extraction and selection
- Conduct model Evaluation

Introduction

The feature of a machine learning technique to categorize or classify an object into its corresponding label with the help of learned discributer from hundreds of image is called as object classification.

This is one of a supervised learning problem where the users must provide training data (set of objects along with its labels) to the machine learning technique so that it learns how to categorize each object (by learning the feature behind) with respect to its class.

In this lab , you will be introduced into one such object classification problem namely malaria detection and classification, which is a hard problem because there is a similarity between the infected and the non infected. As you know machine learning is all about learning from the past data, so huge dataset of malaria images to perform real-time malaria detection and classification.

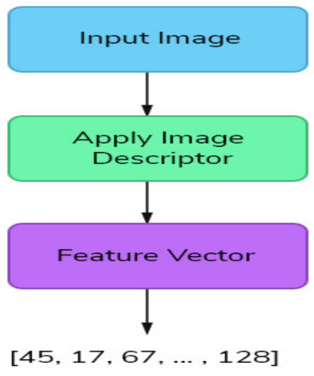
Without caring too much on real-time malaria disease classification, in this lab you will see how to perform a simple image classification task using opencv, and machine learning algorithms with the help of python.

Feature Extraction

Features are the information or list of numbers that are extracted from an image. These are real-valued numbers (integers, float or binary). There are a wider range of feature extraction algorithms in Computer Vision.

When deciding about the features that could quantify infected from non infected, we could possibly think of Color, Texture and Shape as the primary ones. This is an obvious choice to globally quantify and represent the disease infected or non infected.

But this approach is less likely to produce good results, if we choose only one feature vector, as these species have many attributes in common like the non infected will be similar to infected in terms of color and so on. So, we need to quantify the image by combining different feature descriptors so that it describes the image more effectively.



Global Feature Descriptors

These are the feature descriptors that quantifies an image globally. These don't have the concept of interest points and thus, takes in the entire image for processing. Some of the commonly used global feature descriptors are

Color - Color Channel Statistics (Mean, Standard Deviation) and Color Histogram
Shape - Hu Moments, Zernike Moments
Texture - Haralick Texture, Local Binary Patterns (LBP)
Others - Histogram of Oriented Gradients (HOG), Threshold Adjacency Statistics (TAS)

Local Feature Descriptors

These are the feature descriptors that quantifies local regions of an image. Interest points are determined in the entire image and image patches/regions surrounding those interest points are considered for analysis. Some of the commonly used local feature descriptors are

SIFT (Scale Invariant Feature Transform)

SURF (Speeded Up Robust Features)
ORB (Oriented Fast and Rotated BRIEF)
BRIEF (Binary Robust Independed Elementary Features)

Combining Global Features

There are two popular ways to combine these feature vectors. For global feature vectors, we just concatenate each feature vector to form a single global feature vector. This is the approach we will be using in this lab.For local feature vectors as well as combination of global and local feature vectors, we need something called Bag of Visual Words (BOVW). This approach will not be discussed in this lab, but there are lots of resources to learn this technique. Normally, it uses Vocabulary builder, K-Means clustering, Linear SVM, and Td-Idf vectorization.

Dataset

Thanks to the National Institute of Health (NIH) the malaria dataset is available for download from https://ceb.nlm.nih.gov/proj/malaria/cell_images.zip.

The malaria dataset is composed of a total of 27,598 segmented cell images extracted from thin blood smear slide images. The cell images are organized into two folders, parasitized and uninfected, with 13,799 cell images in each, making this a balanced dataset.For more information about the dataset please read the aforementioned link.

Organizing Dataset

The folder structure for this lab is given below.

Create a dataset folder and subfolders Status_Healthy, Status_infected inside it.

Select randomly and copy 80% of parasitized and uninfected cell images into Status_Healthy, Status_infected folders.

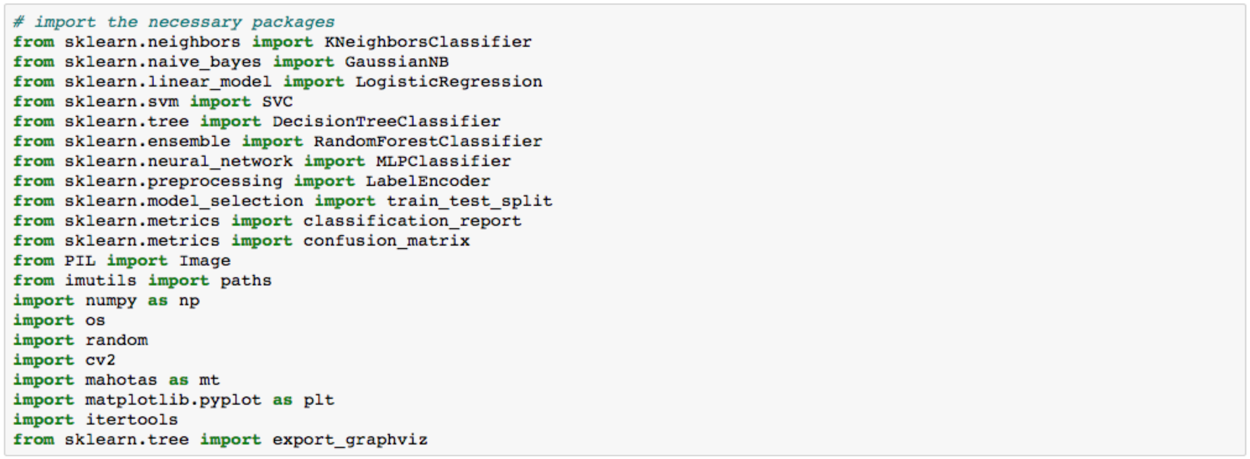
Crate another testdata folder and subfolders Status_Healthy, Status_infected inside it.

Select randomly and copy 20% of the cell images from each category and put them in the testdata folder.

Global Feature Extraction

The three global feature descriptors are

Color Histogram that distinguish color of infected from non infected.
Hu Moments that quantifies shape of infected from non infected.
Haralick Texture that quantifies texture of infected from non infected.
As you might know images are matrices, so there should be an efficient way to store the feature vectors locally. The program takes one image at a time, extract three global features, concatenates the three global features into a single global feature and saves it along with its label in an HDF5 file format.
Instead of using HDF5 file-format, “.csv” file-format could be used to store the features. But, as the amount of data is very large,using HDF5 format is worth it.
Importing Required Libraries
First, the required libraries to carry out the experiment are imported as follows:

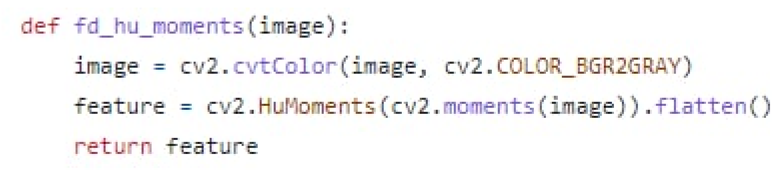


```
# import the necessary packages
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from PIL import Image
from imutils import paths
import numpy as np
import os
import random
import cv2
import mahotas as mt
import matplotlib.pyplot as plt
import itertools
from sklearn.tree import export_graphviz
```

scikit-image, opencv, to read cell image files and process them as required
matplotlib and seaborn to view cell images and plot some graphs
basic python os and math functionality
numpy and random to manipulate arrays and generate random numbers
scikit-learn, sklearn, to carry feature engineering, model fitting and hyperparameter searching
Installation of libraries
To install any library you can use any either conda or pip. For instance to install open cv search pip install opencv on the web and then copy and past in the cmd terminal.
To install the mahotas package use this .
conda install -c conda-forge mahotas

Functions for global feature descriptors

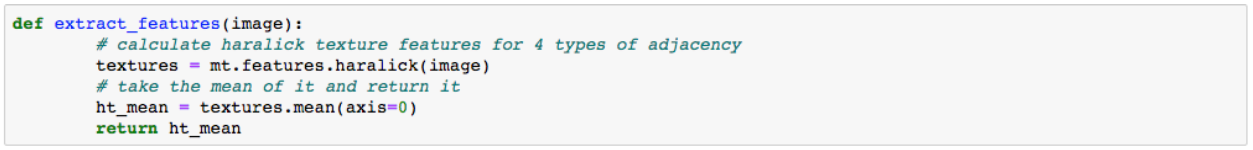
To extract Hu Moments features from the image, cv2.HuMoments() function provided by OpenCV will be used. The argument to this function is the moments of the image cv2.moments() flatenned. This means the moment of the image is computed and converted it to a vector using flatten(). Before doing that, the color image should be converted into a grayscale image as moments expect images to be grayscale.



```
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

Haralick Textures

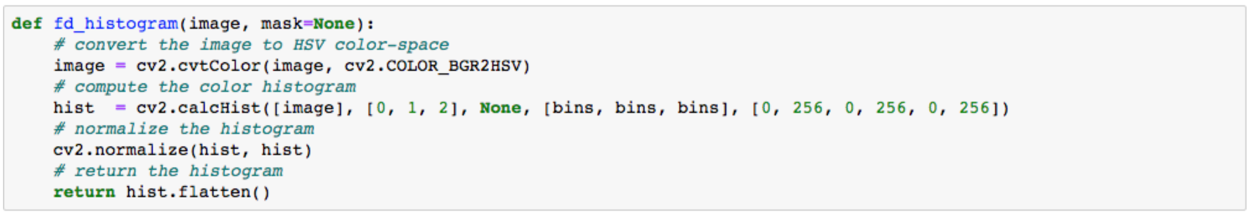
To extract Haralick Texture features from the image, the mahotas library will be used. The function mahotas.features.haralick() will be used. Before doing that, the color image should be converted into a grayscale image as haralick feature descriptor expect images to be grayscale.



```
def extract_features(image):
        # calculate haralick texture features for 4 types of adjacency
        textures = mt.features.haralick(image)
        # take the mean of it and return it
        ht_mean = textures.mean(axis=0)
        return ht_mean
```

Color Histogram

To extract Color Histogram features from the image, the cv2.calcHist() function provided by OpenCV is used. The arguments it expects are the image, channels, mask, histSize (bins) and ranges for each channel [typically 0-256). The histogram is then normalized using normalize() function of OpenCV and return a flattened version of this normalized matrix using flatten().



```
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()
```

For each of the training label name, loop through the corresponding folder to get all the images inside it. For each image, first resize the image into a fixed size. Then, the three global features and concatenate these three features using

NumPy’s np.hstack() function is extracted. Keep track of the feature with its label using those two lists created below - labels and global_features. You could even use a dictionary here. Below is the code snippet to do these.

```
# grab all image paths in the input dataset directory, then initialize
# our list of images and corresponding class labels
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images("dataset")))

random.seed(42)
random.shuffle(imagePaths)
global_feature=[]
data = []
labels = []
IMAGE_DIMS = (50, 50, 1)
bins = 8
```

```
# loop over the input images
for imagePath in imagePaths:
	# load the image, pre-process it, and store it in the data list
	image = cv2.imread(imagePath)
	image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
	fv_histogram  = fd_histogram(image)
	image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
	fv_hu_moments = fd_hu_moments(image)
	features = extract_features(image)
	global_feature = np.hstack([fv_histogram, features, fv_hu_moments])
	data.append(global_feature)
	l = label = imagePath.split(os.path.sep)[-2]
	labels.append(l)
```

After extracting features and concatenating it, the data should be locally saved. Before saving this data, the LabelEncoder() is used to encode the labels in a proper format. This is to make sure that the labels are represented as unique numbers.

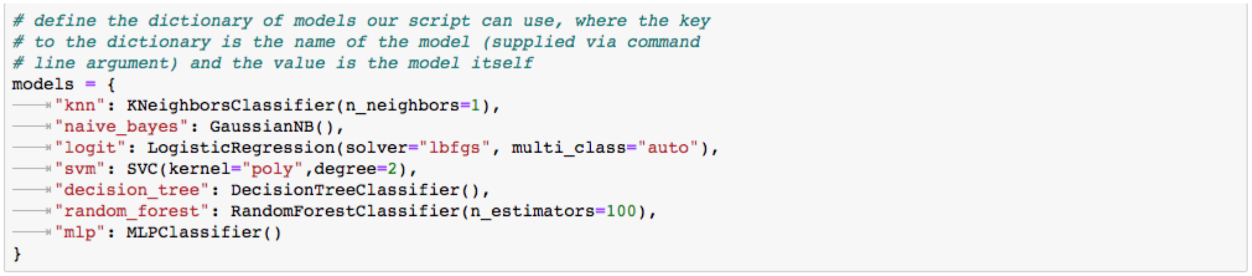
Training classifiers

After extracting, concatenating and saving global features and labels from the training dataset, it’s time to train the system. To do that, the Machine Learning models need to be created. For creating the machine learning model’s, scikit-learn library will be used.

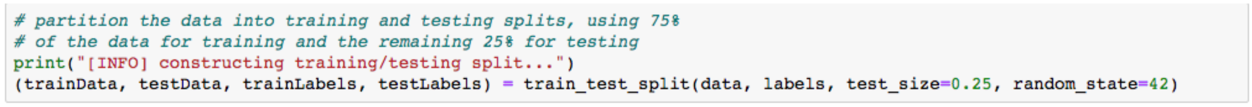
The Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine will be use as the machine learning models. To understand these algorithms, please refer on the internet.

Furthermore, the train_test_split function provided by scikit-learn will be to split the training dataset into train_data and test_data. By this way, the models are trained with the train_data and test the trained model with the unseen test_data. The split size is decided by the test_size parameter.

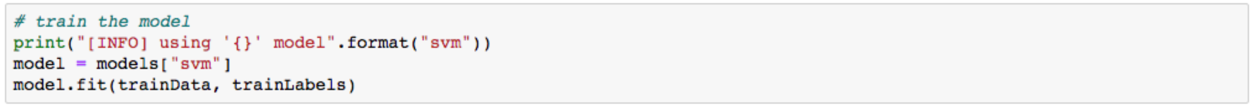
All the necessary libraries to work with are imported and create a models list. This list will have all the machine learning models that will get trained with the locally stored features.



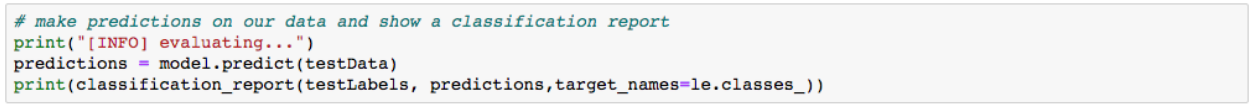
```
# define the dictionary of models our script can use, where the key
# to the dictionary is the name of the model (supplied via command
# line argument) and the value is the model itself
models = {
	"knn": KNeighborsClassifier(n_neighbors=1),
	"naive_bayes": GaussianNB(),
	"logit": LogisticRegression(solver="lbfgs", multi_class="auto"),
	"svm": SVC(kernel="poly",degree=2),
	"decision_tree": DecisionTreeClassifier(),
	"random_forest": RandomForestClassifier(n_estimators=100),
	"mlp": MLPClassifier()
}
```



```
# partition the data into training and testing splits, using 75%
# of the data for training and the remaining 25% for testing
print("[INFO] constructing training/testing split...")
(trainData, testData, trainLabels, testLabels) = train_test_split(data, labels, test_size=0.25, random_state=42)
```



```
# train the model
print("[INFO] using '{}' model".format("svm"))
model = models["svm"]
model.fit(trainData, trainLabels)
```



```
# make predictions on our data and show a classification report
print("[INFO] evaluating...")
predictions = model.predict(testData)
print(classification_report(testLabels, predictions,target_names=le.classes_))
```

Testing classifier

Use the code below to test the model you built.

```
# grab all image paths in the input dataset directory, then initialize
# our list of images and corresponding class labels
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images("testdata")))
random.seed(42)
random.shuffle(imagePaths)
datatest = []
labelstest=[]
IMAGE_DIMS = (50, 50, 1)
# loop over the input images
for imagePath in imagePaths:
    # load the image, pre-process it, and store it in the data list
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    features = extract_features(image)
    datatest.append(features)
    labell = imagePath.split(os.path.sep)[-2]
    labelstest.append(labell)
```

```
# encode the labels, converting them from strings to integers
lb = LabelEncoder()
labels11 = lb.fit_transform(labelstest)
```

```
# make predictions on our data and show a classification report
print("[INFO] evaluating...")
predictionstest = model.predict(datatest)
print(classification_report(labels11, predictionstest,target_names=lb.classes_))
```

```
def plot_confusion_matrix(cm, classes,normalize=False,title='Confusion matrix',cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
cnf_matrix = confusion_matrix(labels11,predictionstest)
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=lb.classes_, title='svm Confusion matrix ')
plt.show()
```

```
conf_matrix=confusion_matrix(testLabels,predictions)
print(conf_matrix)
```

```
import matplotlib.pyplot as plt
import numpy as np
import itertools


def plot_classification_report(classificationReport,
                               title='Classification report',
                               cmap='RdBu'):

    classificationReport = classificationReport.replace('\n\n', '\n')
    classificationReport = classificationReport.replace(' / ', '/')
    lines = classificationReport.split('\n')

    classes, plotMat, support, class_names = [], [], [], []
    for line in lines[1:]:  # if you don't want avg/total result, then change [1:] into [1:-1]
        t = line.strip().split()
        if len(t) < 2:
            continue
        classes.append(t[0])
        v = [float(x) for x in t[1: len(t) - 1]]
        support.append(int(t[-1]))
        class_names.append(t[0])
        plotMat.append(v)

    plotMat = np.array(plotMat)
    xticklabels = ['Precision', 'Recall', 'F1-score']
    yticklabels = ['{0} ({1})'.format(class_names[idx], sup)
                   for idx, sup in enumerate(support)]

    plt.imshow(plotMat, interpolation='nearest', cmap=cmap, aspect='auto')
    plt.title(title)
    plt.colorbar()
    plt.xticks(np.arange(3), xticklabels, rotation=45)
    plt.yticks(np.arange(len(classes)), yticklabels)

    upper_thresh = plotMat.min() + (plotMat.max() - plotMat.min()) / 10 * 8
    lower_thresh = plotMat.min() + (plotMat.max() - plotMat.min()) / 10 * 2
    for i, j in itertools.product(range(plotMat.shape[0]), range(plotMat.shape[1])):
        plt.text(j, i, format(plotMat[i, j], '.2f'),
                 horizontalalignment="center",
                 color="white" if (plotMat[i, j] > upper_thresh or plotMat[i, j] < lower_thresh) else "black")

    plt.ylabel('Metrics')
    plt.xlabel('Classes')
    plt.tight_layout()
```

Assignment

This question will use the malaria dataset once again. Again, create a test set consisting of 1/2 of the data using the rest for training.

1. Save the features in the form of csv format.

2. Fit a randomForest model, decision tree,Knn,logistic regression model,linear discriminant analysis,quadratic discriminant analysis, and Boosting algorithm model to the training data.

3. Predict the labels for the corresponding test data.

4. Compute the confusion matrix for the test data.
5. Compute the AUC (Area Under the Curve) for each classifier.
6. Plot ROC curves as evaluate on the test data.
7. Out of all classifiers used in this assignment, which would you choose as a final model for the malaria data?