IIPCV Project 2

Kacper Reja

February 2020

Contents

1 Introduction

1.1 Project description

Plant or Flower Species Classification is one of the most challenging and difficult problems in Computer Vision due to a variety of reasons. Goal of this project is to find the best possible classification system for set of leaves images of 6 different types using feature vectors and machine learning algorithms. Number of images per each type is from 38 to 97.

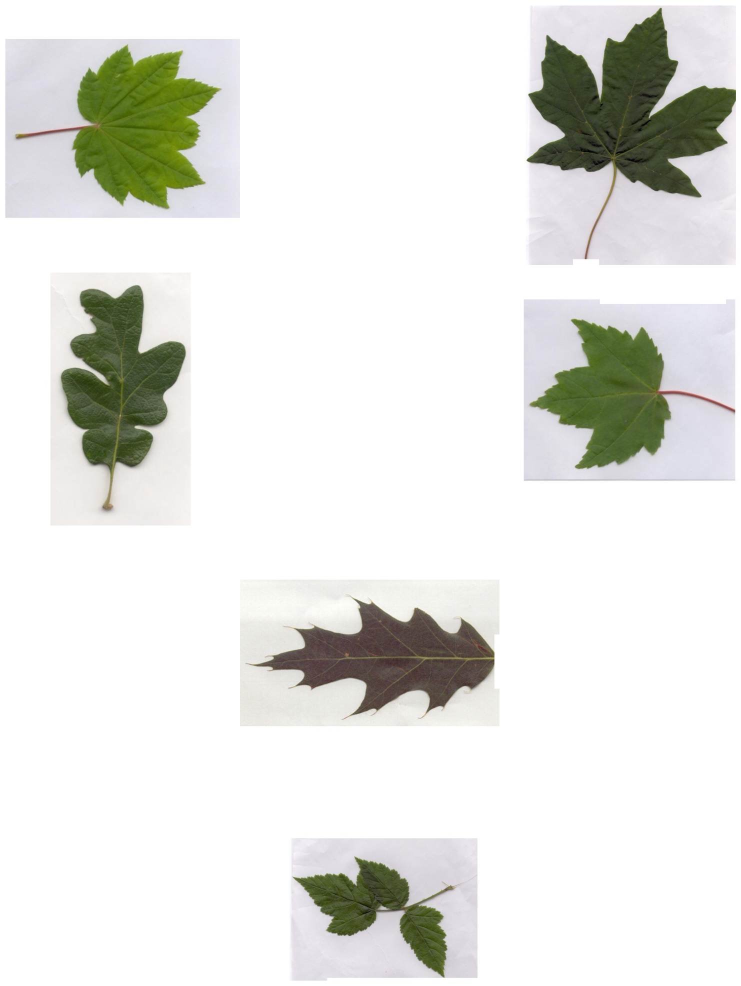


Figure 1: Sample image examples

Classification will be based on feature vectors extracted from images which then will be used by few machine learning algorithms and we will choose which one works the best.

2 Solution description

2.1 Feature extraction

Our first step will be extraction of feature vectors from images and saving them locally in h5 format. Features are the information or list of numbers that are extracted from an image. There are a wider range of feature extraction algorithms in Computer Vision. When deciding about the features that could quantify plants and flowers, we could possibly think of Color, Texture and Shape as the primary ones. This is an obvious choice to globally quantify and represent the plant or flower image. 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. So, we need to quantify the image by combining different feature descriptors so that it describes the image more effectively.

2.1.1 Hu Moments

Shape of leaves will be described using hu moments feature vector. To extract Hu Moments features from the image, we use cv2.HuMoments() function provided by OpenCV. The argument to this function is the moments of the image cv2.moments() flatenned. It means we compute the moments of the image and convert it to a vector using flatten(). Before doing that, we convert our color image into a grayscale image as moments expect images to be grayscale.

```
# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

2.1.2 Haralick Textures

Texture of samples will be described using haralick textures feature vector. To extract Haralick Texture features from the image, we make use of mahotas library. The function we will be using is mahotas.features.haralick(). Before doing that, we convert our color image into a grayscale image as haralick feature descriptor expect images to be grayscale.

```
# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick
```

2.1.3 Color Histogram

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

```
# feature-descriptor-3: Color Histogram
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()
```

2.2 Training machine learning models

The second step will be to train various machine learning models using our feature vectors. We will choose Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Trees, Random Forests, Gaussian Naive Bayes and Support Vector Machine as our machine learning models.

```
# create all the machine learning models
models = []
models.append(('LR', LogisticRegression(random_state=seed)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=seed
   ↪ )))
models.append(('RF', RandomForestClassifier(n_estimators=
   ↪ num_trees, random_state=seed)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=seed)))
```

After that, we split our data into training set and testing set with proportions 80/20.

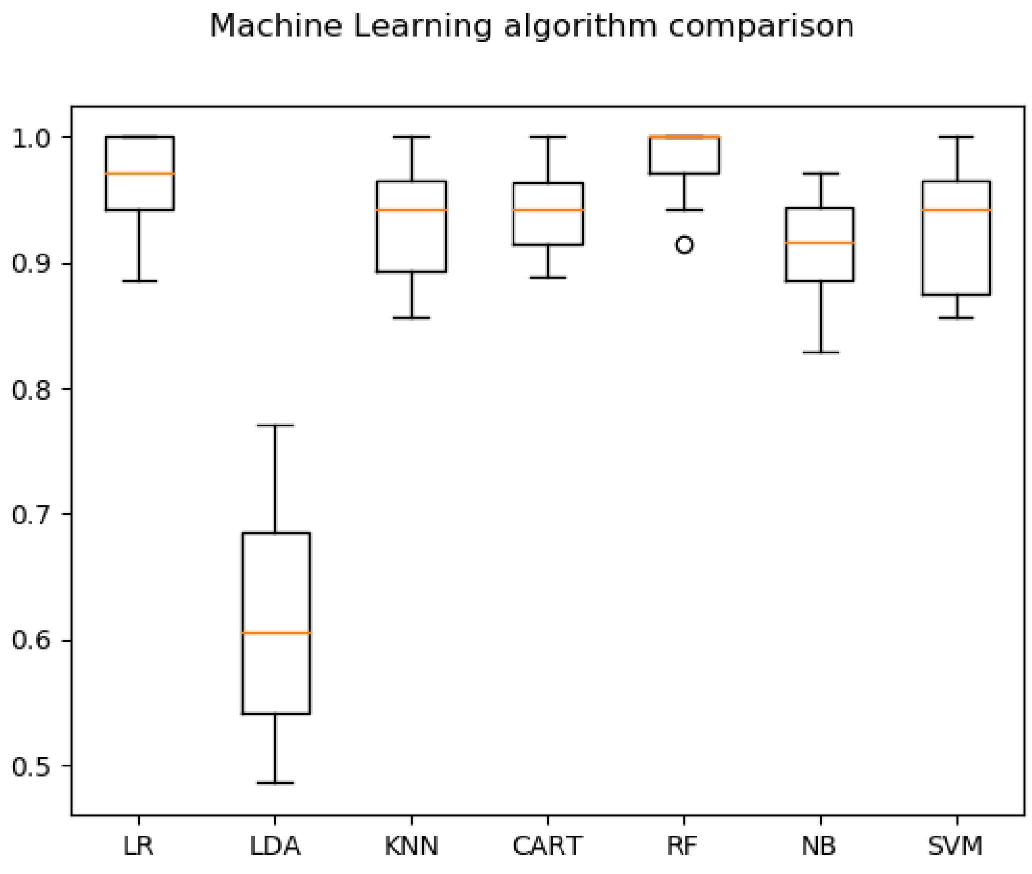
```
# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal,
    ↪ testLabelsGlobal) = train_test_split(np.array(
    ↪ global_features), np.array(global_labels), test_size=
    ↪ test_size, random_state=seed)
```

And finally we perform 10-fold cross validation on our training models.

```
# 10-fold cross validation
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, trainDataGlobal,
        ↪ trainLabelsGlobal, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s:␣%f␣(%f)" % (name, cv_results.mean(), cv_results.
        ↪ std())
    print(msg)
```

3 Results

From the code given above we get following results:
LR: 0.963095 (0.038144)
LDA: 0.617937 (0.089415)
KNN: 0.929206 (0.046037)
CART: 0.937857 (0.034859)
RF: 0.980079 (0.028691)
NB: 0.909206 (0.049332)
SVM: 0.926349 (0.049410)
And following boxplot algorithm comparison:



We can see that almost all of the machine learning algorithms that we used had high accuracy above 90% and only Linear Discriminant Analysis had bad accuracy of 60%. The best algorithm for our task turned out to be Random Forest algorithm with accuracy of 98% with only 1 outlier causing problems.

4 Summary

After result analysis we can see that combination of Hu Moments, Haralick Texture and Color Histogram feature vectors provided us with very high accuracies for majority of machine learning algorithms that we used. Perhaps using an additional feature vector could yield accuracies at the level of 100% for some of the best algorithms that we used.

5 Source code

main.py

```
#-----------------------------------

# GLOBAL FEATURE EXTRACTION

#-----------------------------------

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler

from matplotlib import pyplot

import numpy as np

import mahotas

import cv2

import os

import h5py
```

```
#--------------------
# tunable-parameters
#--------------------
fixed_size = tuple((500, 500))
train_path = "dataset/train"
h5_data = 'output/data.h5'
h5_labels = 'output/labels.h5'
bins = 8

# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick
```

```
# feature-descriptor-3: Color Histogram
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()

# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)

# empty lists to hold feature vectors and labels
global_features = []
labels = []

# loop over the training data sub-folders
for training_name in train_labels:
```

```
# join the training data path and each species training
    ↪ folder
dir = str(train_path) + "/" + str(training_name)

# get the current training label
current_label = training_name

# loop over the images in each sub-folder
for x in os.listdir(dir):
    # get the image file name
    file = dir + "/" + str(x)

    # read the image and resize it to a fixed-size
    image = cv2.imread(file)
    image = cv2.resize(image, fixed_size)

     ####################################
    # Global Feature extraction
    ####################################
    fv_hu_moments = fd_hu_moments(image)
    fv_haralick = fd_haralick(image)
    fv_histogram = fd_histogram(image)
    ###################################
    # Concatenate global features
    ###################################
```

```
        global_feature = np.hstack([fv_histogram, fv_haralick,
            ↪ fv_hu_moments])

        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)

    print("[STATUS]␣processed␣folder:␣{}".format(current_label)
        ↪ )

print("[STATUS]␣completed␣Global␣Feature␣Extraction...")

# get the overall feature vector size
print("[STATUS]␣feature␣vector␣size␣{}".format(np.array(
    ↪ global_features).shape))

# get the overall training label size
print("[STATUS]␣training␣Labels␣{}".format(np.array(labels).
    ↪ shape))

# encode the target labels
targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)
print("[STATUS]␣training␣labels␣encoded...")
```

```
# scale features in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)
print("[STATUS]␣feature␣vector␣normalized...")

print("[STATUS]␣target␣labels:␣{}".format(target))
print("[STATUS]␣target␣labels␣shape:␣{}".format(target.shape))

# save the feature vector using HDF5
h5f_data = h5py.File(h5_data, 'w')
h5f_data.create_dataset('dataset_1', data=np.array(
   ↪ rescaled_features))

h5f_label = h5py.File(h5_labels, 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))

h5f_data.close()
h5f_label.close()

print("[STATUS]␣end␣of␣training..")
```

train_test.py

```
#-----------------------------------
# TRAINING OUR MODEL
#-----------------------------------
import h5py
import numpy as np
import os
import glob
import cv2
import warnings
from matplotlib import pyplot
from sklearn.model_selection import train_test_split,
   ↪ cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score,
   ↪ classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import
   ↪ LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.externals import joblib
```

```
warnings.filterwarnings('ignore')

#--------------------
# tunable-parameters
#--------------------
num_trees = 100
test_size = 0.20
seed = 9
train_path = "dataset/train"
test_path = "dataset/test"
h5_data = 'output/data.h5'
h5_labels = 'output/labels.h5'
scoring = "accuracy"

# get the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()

if not os.path.exists(test_path):
    os.makedirs(test_path)

# create all the machine learning models
```

```
models = []
models.append(('LR', LogisticRegression(random_state=seed)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=seed
    ↪ )))
models.append(('RF', RandomForestClassifier(n_estimators=
    ↪ num_trees, random_state=seed)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=seed)))

# variables to hold the results and names
results = []
names = []

# import the feature vector and trained labels
h5f_data = h5py.File(h5_data, 'r')
h5f_label = h5py.File(h5_labels, 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string = h5f_label['dataset_1']

global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)

```

```
h5f_data.close()
h5f_label.close()

# verify the shape of the feature vector and labels
print("[STATUS]␣features␣shape:␣{}".format(global_features.
    ↪ shape))
print("[STATUS]␣labels␣shape:␣{}".format(global_labels.shape))

print("[STATUS]␣training␣started...")

# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal,
    ↪ testLabelsGlobal) = train_test_split(np.array(
    ↪ global_features), np.array(global_labels),test_size=
    ↪ test_size, random_state=seed)

print("[STATUS]␣splitted␣train␣and␣test␣data...")
print("Train␣data␣␣:␣{}".format(trainDataGlobal.shape))
print("Test␣data␣␣␣:␣{}".format(testDataGlobal.shape))
print("Train␣labels:␣{}".format(trainLabelsGlobal.shape))
print("Test␣labels␣:␣{}".format(testLabelsGlobal.shape))

# 10-fold cross validation
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, trainDataGlobal,
        ↪ trainLabelsGlobal, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s:␣%f␣(%f)" % (name, cv_results.mean(), cv_results.
        ↪ std())
    print(msg)

# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Machine␣Learning␣algorithm␣comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

References

[1] https://gogul.dev/software/image-classification-python?fbclid=IwAR2iOYYM3ecUNOfBeqhvKgNrjeQh3jFlhNmzDIf8P9UgwaShZFNJDxOXrfE

[2] https://en.wikipedia.org/wiki/Random_forest

[3] https://en.wikipedia.org/wiki/Image_moment

[4] https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/