INTRODUCTION TO IMAGE PROCESSING AND COMPUTER VISION

Project 2: Plant species recognition

Szymon Majorek

s.majorek@student.mini.pw.edu.pl

Table of contents

1. Introduction

1) General Introduction

Machine learning is a study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instruction, relying on patterns and inference instead. In other words, we can teach program to do something for us, without us giving explicit input for our decision. In our case we will be using Random forests model.

Image recognition on the other hand is the ability to identify and detect objects or features of digital images or videos. It is a method for capturing, processing, examining and sympathizing images. We can use machine learning for image recognition, as we will in this project.

The goal of this project is to use machine learning in order to recognize different species of plants based on images of their leaves. We aim to teach our program to recognize different plant species by teaching it how do their leaves look like.

2) Data set presentation

Our data set is quite simple actually. We have 6 types of leaves with irregular number of images of them. The types are as follows*: Acer Circinatum, Acer Glabrum, Acer Macrophyllum, Acer Negundo, Quercus Garryana* and *Quercus Kelloggii*:

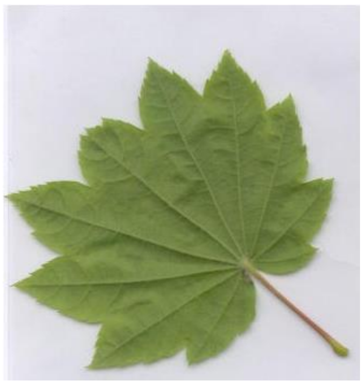


Figure 1 - Acer Circinatum

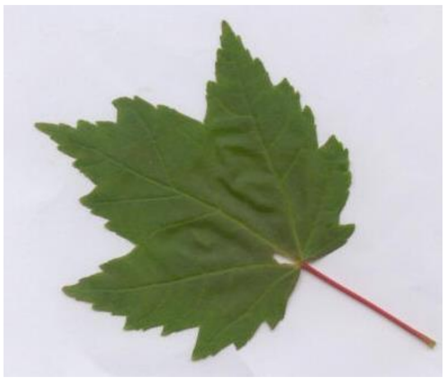


Figure 2 - Acer Glabrum

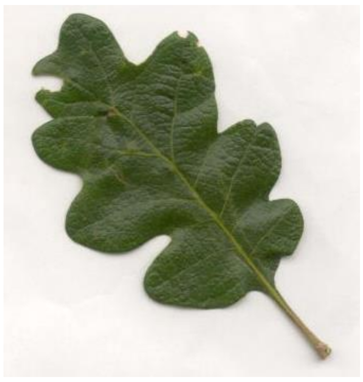


Figure 3 - Quercus Garryana

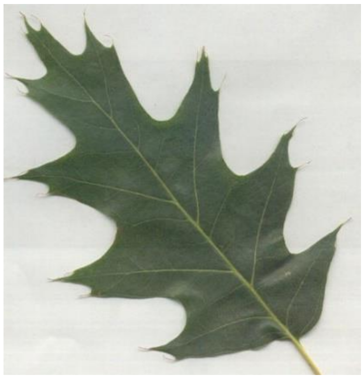


Figure 4 - Quercus Kelloggii

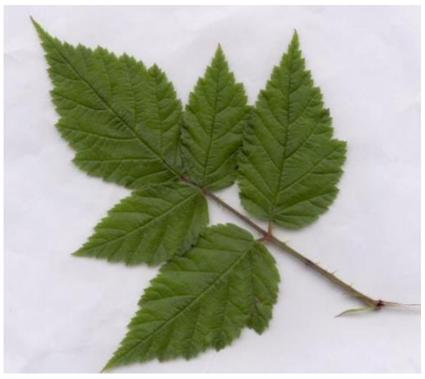


Figure 5 - Acer Negundo

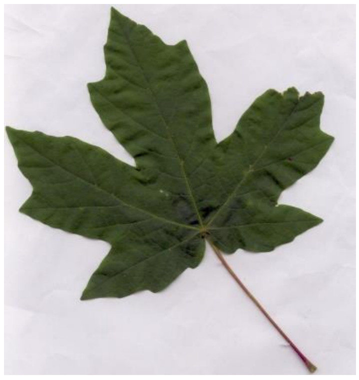


Figure 6 - Acer Macrophyllum

When looking at those images it's quite easy to say that they are all different species. For example the one on fig 3 is sort of compact but with rounded tips, which is different from fig. 4 which has pointy tips instead. In fact all of them have pointy tips apart from the 3rd one however they have some slight differences which are easy to notice a first glance. They even have different colours but that might just be a coincidence that I randomly chose those examples which actually have visibly different colours. Now that is all obvious by looking at the pictures, however in terms of our project, we need to think a little differently. We know what are leaves, we know how they look like and where to look for differences between leaves of different plants (shape, number of pointy ends, whether the ends are actually pointy or more rounded etc.). One noticeable thing that us, humans don't usually look for when trying to distinguish different leaves is their colour because we tend to think of leaves as generally green (well with small variations depending on seasons).

2. Feature extraction

1) Short introduction

Features are the information or list of numbers that are extracted from an image. These are real-valued numbers. As mentioned in the introduction, we need to think about what we want to actually find, as we probably won't be able to categorize a leaf by it's number of veins or width of it's stem. We could settle on just one feature however as we can see above some of the leaves are rather similar so it's better to use more than one feature to try to find their classes.

2) Features and potential drawbacks

We choose to look at texture (haralick, local binary pattern), color (histogram), shape (moments). Even though we tend to think of leaves as just green, it might be worth it to use color histogram in order to categorize those leaves, after all those 6 picked example do actually vary in color slightly, so why not try to use it. Example:

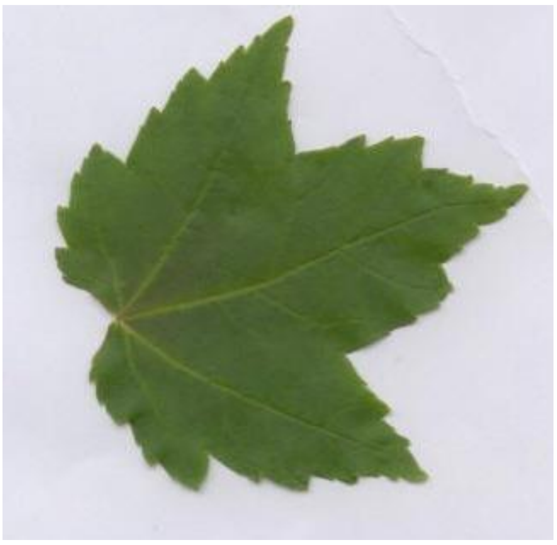


Figure 7 - Acer Glabrum example

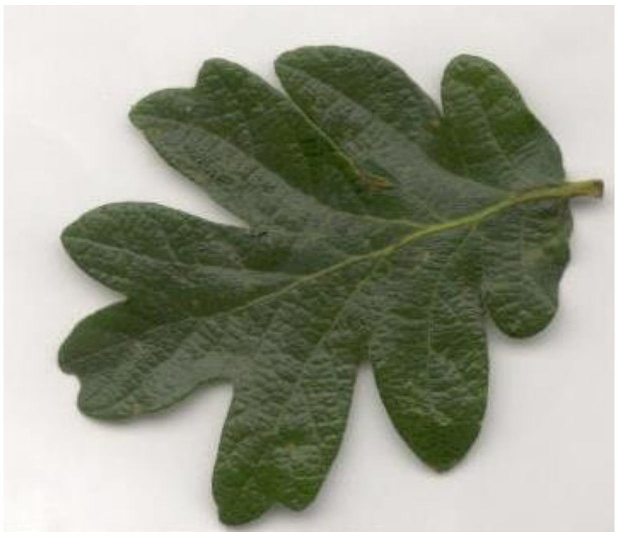


Figure 8 - Quercus Garryana

Here we can actually see that the difference in color is quite visible and obvious. Well maybe not the color itself but rather the shade, and the fact that Garryana leaf is much darker in tone. They are additionally quite different in texture, as the one on left appear to be much smoother judging by this image.

However, as usually, it might not be the best choice afterall. We can take a look at few examples that would steer us away from choosing this feature, as some classes, for example:

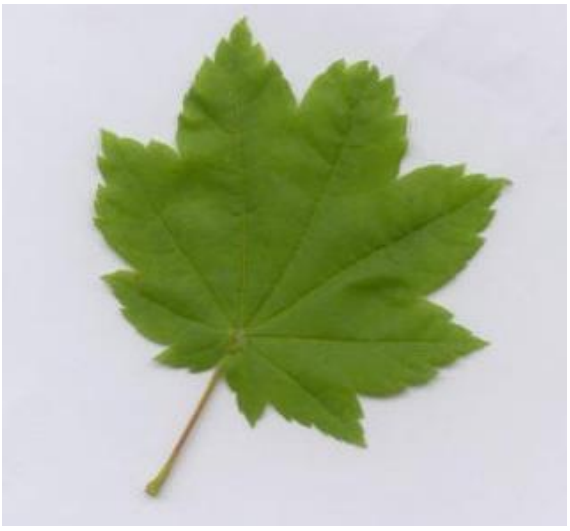


Figure 9 - Acer Circinatum example

For example if we add this leaf to our consideration we see that it’s much more similar to Acer Glabrum than the Quercus Garryana leaf. Actually it’s very similar in both categories, colour, as well as shape. The texture also doesn’t appear to be vastly different, with the slight difference being the veins, in Circinatum leaf they appear to a little convex, whereas in Glabrum they’re more flat.

But the problems may arise not only between different classes of leaves. They can just as well arise between leaves of theoretically the same class.

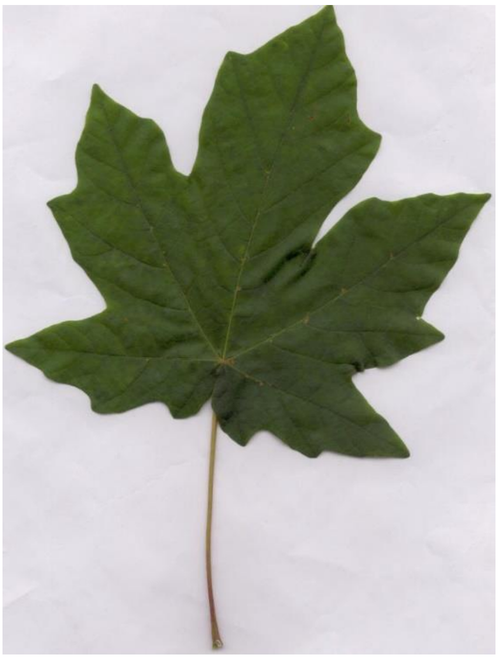


Figure 10 - Macrophyllum ex 1

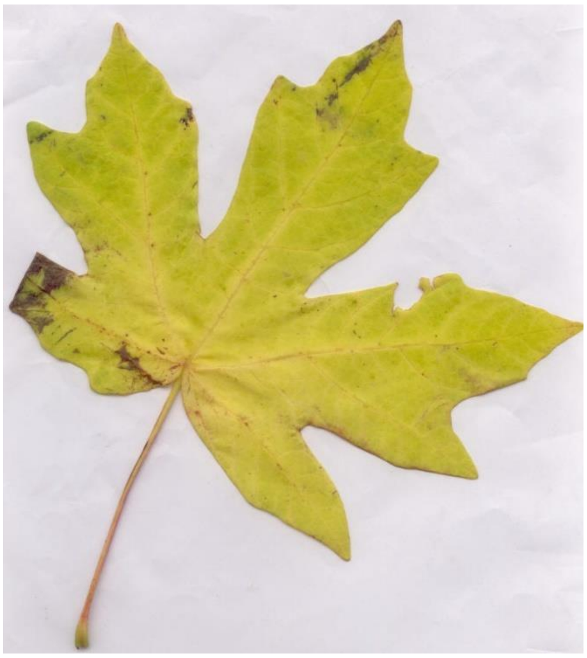


Figure 11 - Macrophyllum ex 2

Those 2 leaves we see above are both from the same category – Acer Macrophyllum. The difference between them is extremely vivid and we can immediately pin point it to the fact that the leaf from figure 11 was picked up during fall, and not summer or spring. For us, humans, it’s fairly obvious. Thankfully for I’ve picked rather extreme case for this example as this takes place only once in this class so we can still expect fairly good results.

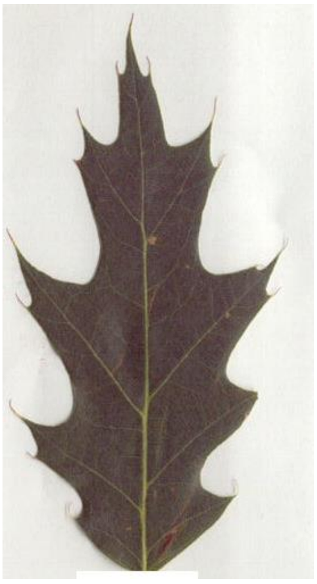


Figure 12 - Kelloggii ex 1

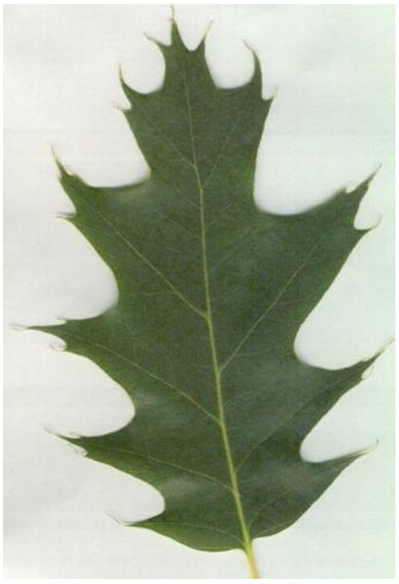


Figure 13 - Kelloggii ex 2

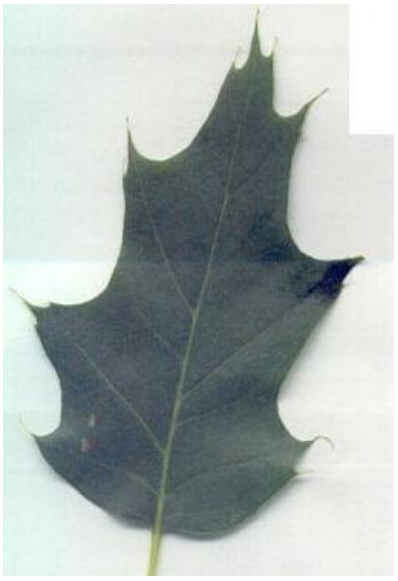


Figure 14 - Kelloggii ex 3

These 3 images are all of the same class of leaf – Quercus Kelloggii. As we can clearly see all of 3 those example have different colour. It's not even fair to say that it's a slight difference. The colours are nothing like each other, one is blue, one is green and the last one is very dark red. The situation would be similar as before, however in this case the situation keeps repeating, so inside this class of leaves we have few classes of colours, hence we should expect our results when using colour histogram to be slightly worse compared to other classes of leaves.

Having considered colour, we can move onto the next feature we will be considering, namely texture. We've already briefly mentioned in when discussing colour, but I'd like to expand a little on that.

Image texture in image processing is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of colour or intensities in an image or selected region of an image.

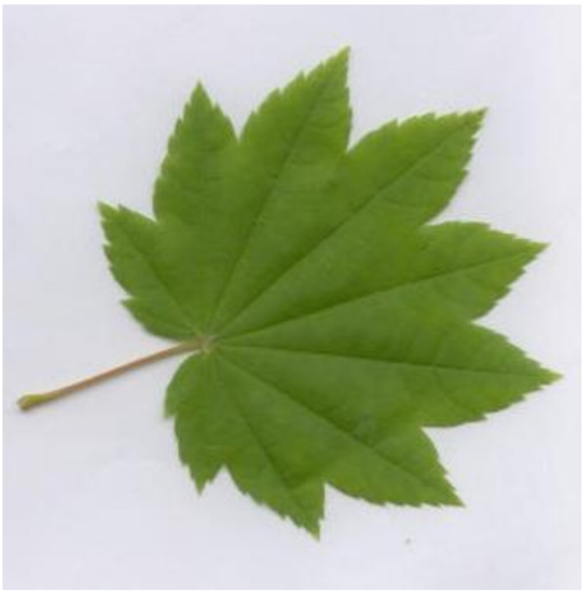


Figure 15 - Circinatum ex

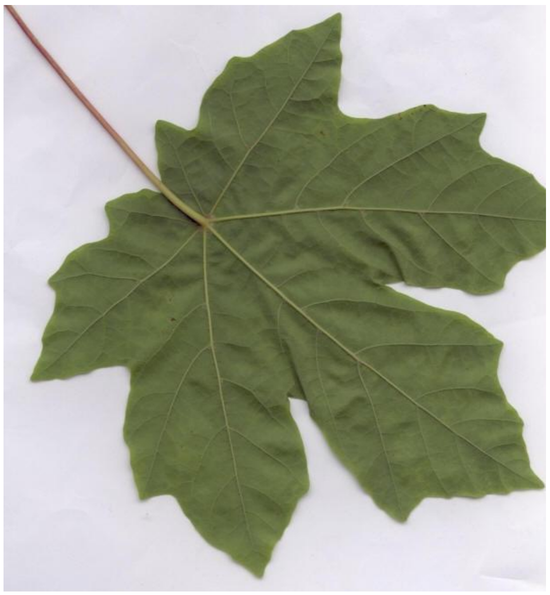


Figure 16 - Macrophyllum

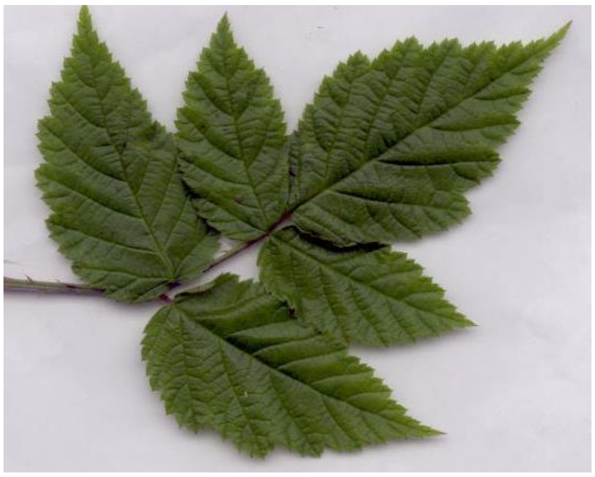


Figure 17 - Negundo ex

Going from figure 15 to 16 we can notice gradual change in the texture. Image presented on figure 15 Is clearly the one with the smoothest surface whereas figure 17 presents heavily wrinkled leaf. With each vein leaving visible mark on the leaf. Of course those example are rather cherry picked just to show that there are actual differences in texture and using this feature actually makes some sense. There’s one more feature that will be considered here – shape. We will be considering shape, however I’m not actually that optimistic about it, I think that looking at shape may turn out to be not as good as using other features even though there’s a clear distinction between some of the classes of leaves. We can take a look at the following comparisons:

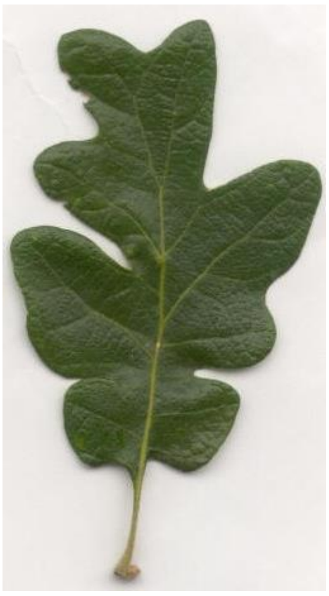


Figure 18 - Garryana ex

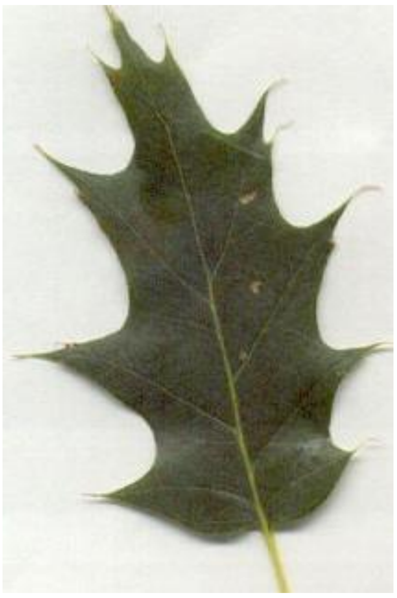


Figure 19 - Kelloggii ex

Here we have images of 2 different classes which look fairly similar. They both have big, solid body in sort of rectangular shape, with few parts sticking out. The difference being that in fig. 18 example the parts that are sticking out are much more round compared to figure 19 where they’re spiky. It could lead to some false positives during training and then later on testing the method.

If we were to look at just the shape feature then definitely these 2 examples – namely Circinatum from figure 20 as well as globrum (figure 21). I think that those two are much more similar than they are different, which in turn means that comparing them by their shape might not be the greatest idea. Having discussed features, we can actually proceed to the way we get them.

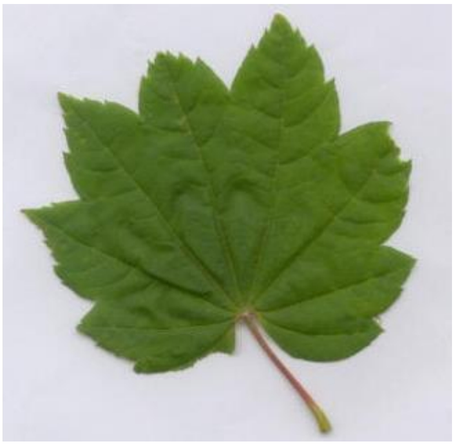


Figure 20 - Circinatum

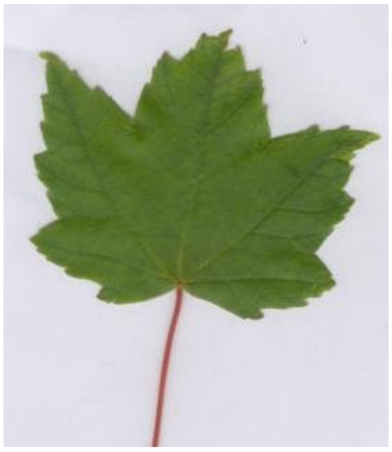
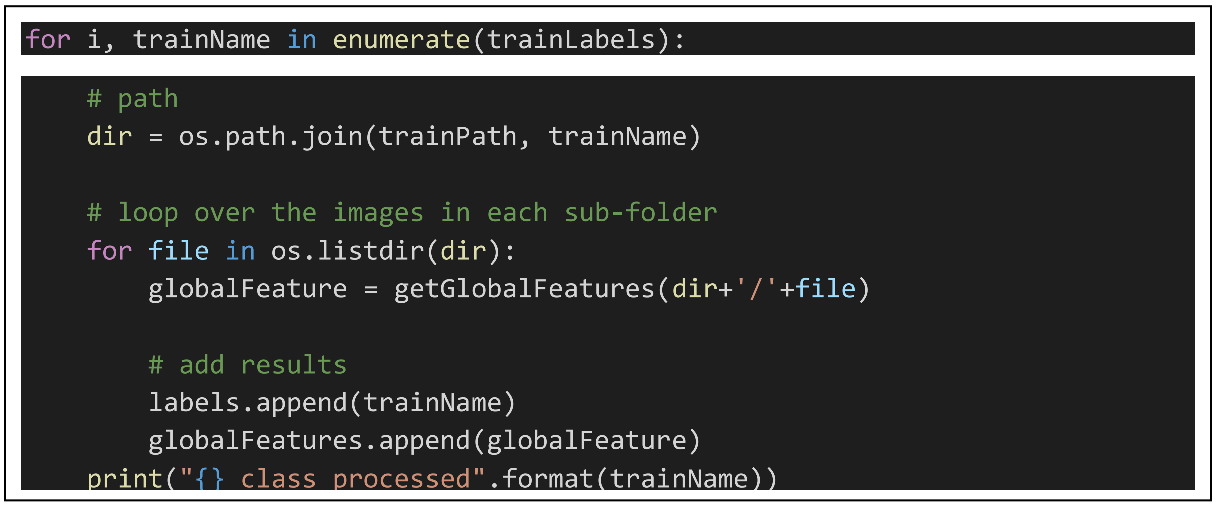


Figure 21 - Glabrum

3) Algoritgm

Even though we have two .py scripts for extracting features they are basically the same and they only differ in the sense that one of them is meant for testing the whole data set using cross validation, whereas the other is meant for 80-20 split of our dataset (80-20 meaning that we take 80% of our data and provide it as training part, and then we test our results using the remaining 20%).

Apart from opening, closing, writing to and from files, the most important thing is obviously the extraction of features from given directory or file.



```
for i, trainName in enumerate(trainLabels):

    # path
    dir = os.path.join(trainPath, trainName)

    # loop over the images in each sub-folder
    for file in os.listdir(dir):
        globalFeature = getGlobalFeatures(dir+'/'+file)

        # add results
        labels.append(trainName)
        globalFeatures.append(globalFeature)
    print("{} class processed".format(trainName))
```

Figure 22 - training data set

This is the core behind features extraction in our global_features*.py files. We iterate over all our train labels in some directory and then over all the files in that directory. In this part of the code there's one function which is probably the most important one in the whole code. The getGlobalFeatures. This is a function defined in helper.py file. Along with that there are few more important ones, namely:

- *fu_hu_moments(image)* -> responsible for shape
- *fd_haralick(image)* -> responsible for texture
- *fd_histogram(image)* -> responsible for colour
- *fd_localBinaryPatterns(image, numPoints = 24, radius = 8)* -> responsible for texture

When acquiring results, we can simply comment out whatever's not the point of their interest as well as remove it from np.hstack method right before returning final result.

After our feature vectors are calculated, we can save them to some file. Since we're dealing with datasets I used .h5 file which seem to be handling datasets very well, and additionally they're pretty easy to deal with.

In fact the algorithms for 80-20 split and whole dataset are identical, they only differ by paths that we use in order to get to the images, and eventually output, depending on what we want to actually do.

3. Testing obtained data

1) Introduction

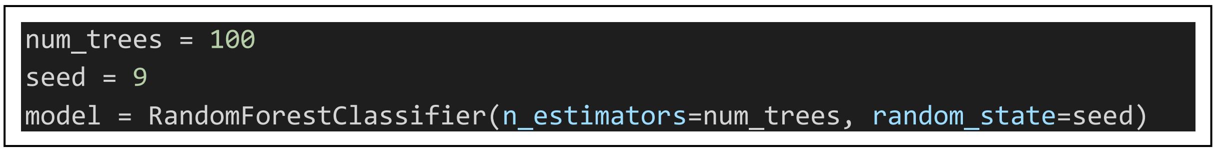
Now that we have our features extracted and saved to files we can proceed to training a model and then validating said model, or just generally checking it's success rate for some test sample.

The whole testing process occurs in train_test.py. First we declare all the paths, arrays, etc. so as to make our lives easier, and then we proceed:

We're using *scikit-learn* library for python for our machine learning models and testing.

The model we're using is *Random Forest*. It is an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of classes (classification) or mean prediction (regression) of the individual trees.

Ensemble here means that it combines more than one algorithm of the same or different kind for classifying objects. Our *Rando Forest Clssifier* creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to dcide the final class of the test object. In other words, it predicts based on the majority of votes from each of the decision trees made, thus reduction the effect of noise.



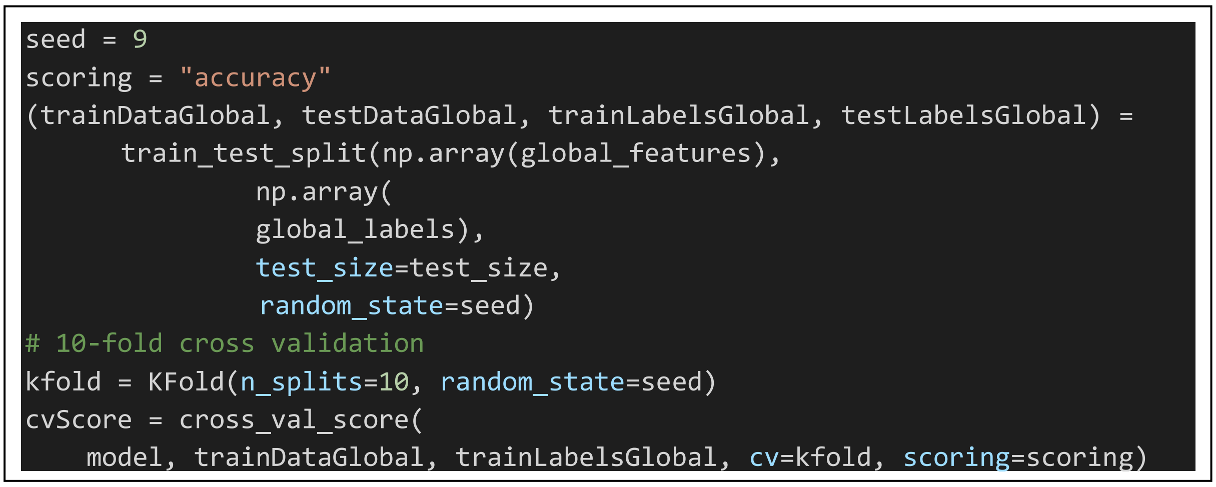
```
num_trees = 100
seed = 9
model = RandomForestClassifier(n_estimators=num_trees, random_state=seed)
```

Figure 23 - model initiation

Parameters we use:

- *n_estimators* – number of trees in the forest.
- *random_state* – it controls the randomness of boostrapping of the samples used when building trees

2) Cross validation



```
seed = 9
scoring = "accuracy"
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) =
      train_test_split(np.array(global_features),
               np.array(
               global_labels),
               test_size=test_size,
               random_state=seed)
# 10-fold cross validation
kfold = KFold(n_splits=10, random_state=seed)
cvScore = cross_val_score(
    model, trainDataGlobal, trainLabelsGlobal, cv=kfold, scoring=scoring)
```

Figure 24 - KFold testing

train_test_split – Used to split our data into random train and test subsets of it. The first two parameters are simply data we obtained and then

- *test_size* – a float between 0 and 1 representing the proportion of the dataset to include in the test split.
- *random_state* – seed used by the random number generator

In this case, we again want to not only print the results to the terminal, but to also store them somewhere on the actual disk. Again I chose .h5 file for this purpose.

KFold – cross-validator. Itprovides train/test indices to split data in train/test sets. Split dataset into k consecutive folds and then each one of them is used as a validation while k-1 remaining folds form the training set. Random_state is the same as before.

cross_val_score – evaluates a score using cross-validation. It returns an array of scores of given estimator for each run of the cross-validation.

- *estimator* – the model we've created before, Random Forest Classifier
- *cv* – determines the cross-validation splitting strategy. In our case we provide the one created in the line above – using KFold function.
- *scoring* – a string indicating what method should be used to get a score, here we pass "accuracy".

The value returned by *cross_val_score* is the result we're interested in. We use .h5 files to save it so that we can process it later. (More about it in the results section)

3) 80-20 split

Here the situation is slightly different. We want to use a rigid split instead of a random one. To achieve that we split our dataset in 4/1 proportion (80% training to 20% testing). We split it by simply taking first 80% of images as the training set and the remaining part as testing set.

```
model.fit(global_features_split, global_labels_split)
```

Figure 25 - 80-20 method, model init

Since we did the split manually (putting training and testing images into different directories), we fit the whole data read from directory to the model (labels are obviously the same).

```
global_feature = getGlobalFeatures(dir+"/"+file)
prediction = model.predict([global_feature])[0]
resultSplit = testName == testLabels[prediction]
```

Figure 26 - Single image testing

Once we have our model, we basically loop through all our test images, we use the *getGlobalFeatures* function from *helper.py*. We use our model to get a prediction for each image using *predict* function. It returns the predicted class based on the vote of trees in the forest with weighted probability estimates. Which means that the returned prediction is the one with highest mean probability estimate across the trees.

Having done that, we save our results once again to process them later, however this time we use .csv files as I've felt like they were easier to manage due to the fact that our script is written in a way that forces us to run it for given method (or set of methods) separately.

4. Results

Results are processed by *results.py* script which creates *results.txt* file in which we can find all the results in a simple text format. Well not exactly all, the result arrays from cross-validation are not there, only their means.

To remind ourselves, we were interested in how well we were able to classify leaves (as in "guess" their species.

1) Cross-validation

The scores here are actually outputs of *cross_val_score* function which are actually arrays. We're mostly interested in mean of them since this is a fine metric to go by when comparing different methods, however since we already have results in the form of arrays, we can actually compare them using boxplots, even though each array has length 10, which is not a lot for a boxplot, it could give us some overview about the obtained results.

Moments (shape)	0.206
Colour histogram	0.98
Haralick texture	0.836
Local binary patterns	0.851
All features combined	0.982

Figure 27 - Features comparison (means)

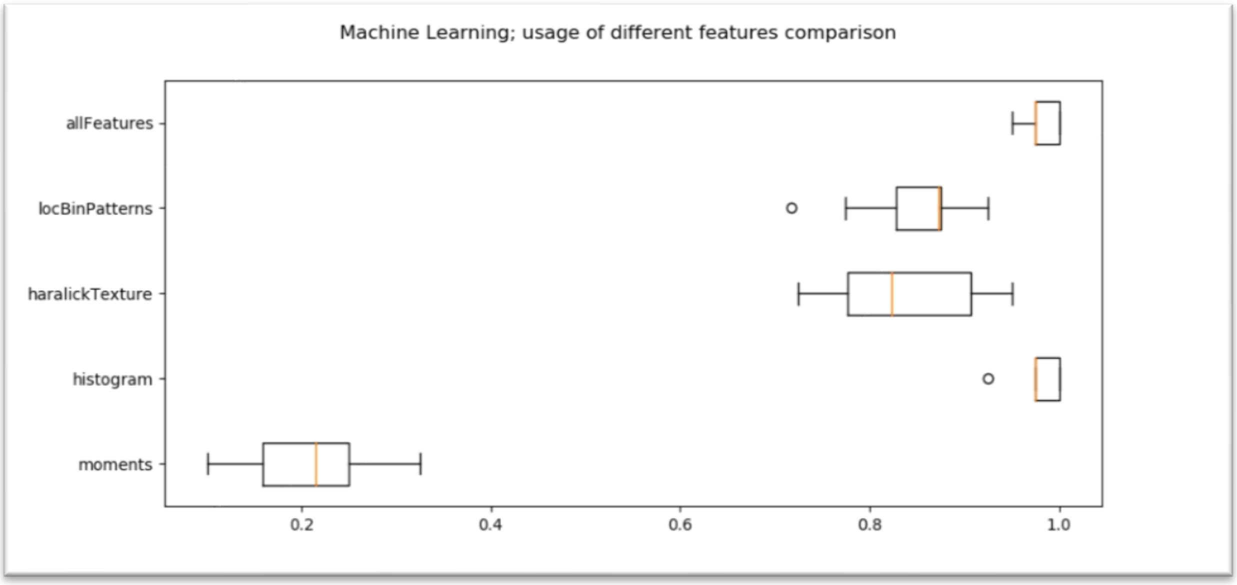


Figure 28 - Features comparison (boxplots)

The first thing that we immediately notice is that the boxplot representing results obtained via *moments* method is very, very far to the left compared to the other methods. This, in very short words, means that our attempt to classify leaves based on their shape failed horribly. We can also tell by the boxplot that it's not very consistent in a sense. I mean that the average score is 0.206 (fig. 27), however the min and max whiskers go from around 0.08 to 0.34 which gives a pretty big range, especially considering other plots. Now it's important to note that it doesn't necessarily mean that the method is terrible. It does however mean that it is terrible in this case, or at least that it wasn't implemented suitably to the task. We can go back a little bit to our discussion about features. For example if we take a look at figures 20 and 21 we see that the differences in shape are rather subtle, and they'll be consistent throughout the whole 2 classes, which could lead to some incorrect results. I'm fairly sure that the method could be vastly improved since the leaves, even though they have slight differences which can be pin pointed, are actually noticeably different in shape. The task would be to write code good enough to pick up on those small differences, which we obviously haven't achieved here.

Having discussed the shape feature, we can throw out its histogram so as to have a more detailed look at the rest of the results.

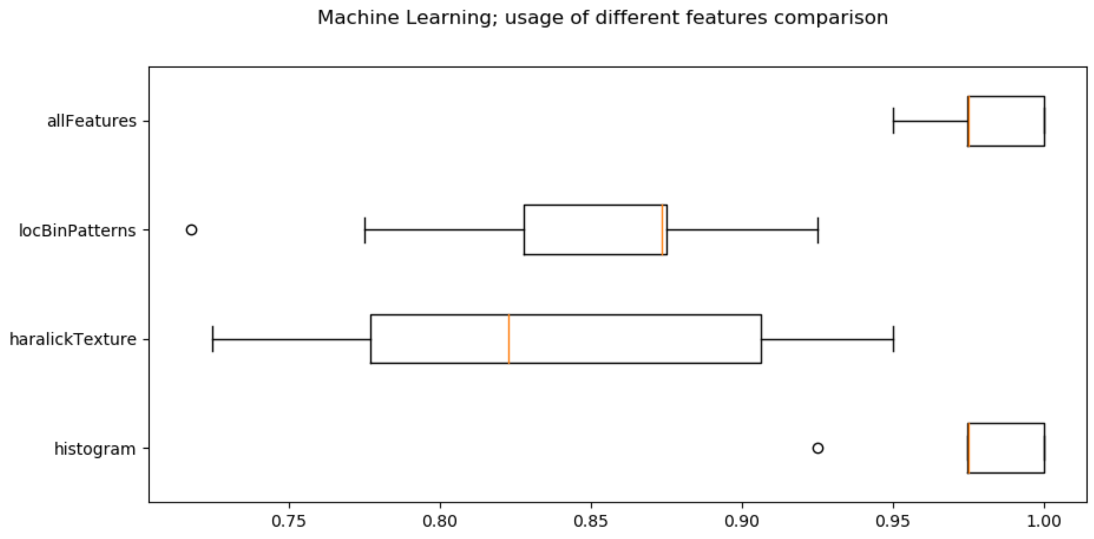


Figure 1 - Features comparison (boxplots, no moments method)

Immediately we notice clearly the best results – colour histogram. Before that though, let's take a look at the other 2 boxplots, namely locBinPatters (Local binary patterns) and haralickTexture. Both of them were using textures and well we can sort of notice that they do indeed have fairly similar shapes and ranges. Haralicktexture has a very wide range, akin to moments boxplot. This would also suggest a small inconsistency in results, although it's mean (0.836) is much greater. Compared mean for methods using textures is 0.8435 which, I'd say, is very good. It predicted correctly nearly all the leaves.

That leaves us with one final plot to discuss – the colour histogram. It is undeniably method that achieved the best results. Not only is it's mean the highest, it is also extremely consistent with only one outlier (0.925 score), which by the way is still almost higher than any other result for all other features. I would say that it's very counter intuitive for a human to see that the best way to categorize leaves was by colour. I'd assume that the one outlier probably has something to do with one of the more extreme cases showcased in previous section.

2) 80-20 split

i) Results

In this section we will focus on the leaf by leaf comparison in addition to method comparisons.

name	circinatum	garryana	glabrum	kelloggii	macrophyllum	negundo	mean
moments	0.077	0.059	0.133	0.263	0.25	0.375	0.193
histogram	1.0	1.0	1.0	0.895	1.0	1.0	0.982
haralick	0.692	1.0	0.667	0.842	0.75	0.875	0.804
locBinPatters	0.769	0.941	0.6	1.0	0.812	0.75	0.812
allFeatures	1.0	1.0	1.0	0.947	1.0	1.0	0.991

Figure 29 - 80-20 split; results by leaf

The story here is similar – histogram is vastly the best, moments the worst and texture is much better than moments, and slightly worse than histogram. However this comparison allows us to analyze the results in a more detailed way; leaf type by leaf type. We can actually see how different features work for different leaves.

ii) Thoughts

For example when using shape feature, we achieved less than 10% success rate for both Circinatum and Garryana classes and a little more than 10% for Glabrum, whereas reslts for Negundo were fairly decent, at almost 38%. If we take a look back at those leaves (figs. 1, 3, 2, 5) we notice that Circinatum and Glabrum are pretty similar so it's not surprising that shape feature wasn't the best for them. In the case of Garryana:

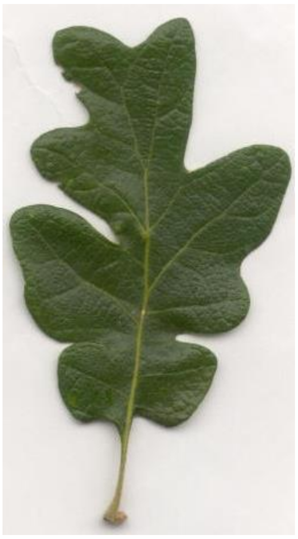


Figure 30 - Garryana ex

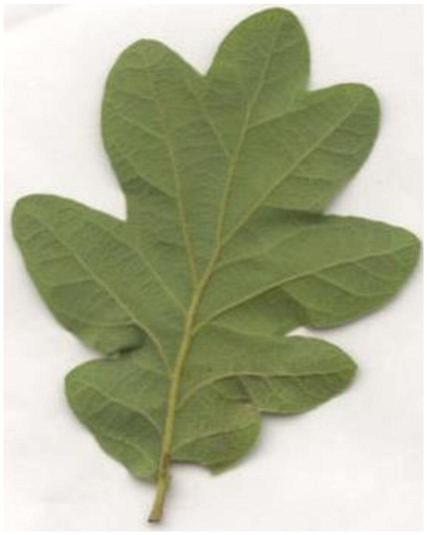


Figure 31 - Garryana ex

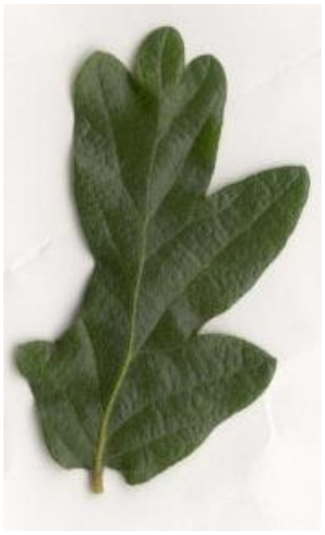


Figure 32 - Garryana ex

We see that even though they belong to the same class their shapes aren't that consistent which probably caused the results to be not the best as well.

Hisogram is actually nearly perfect for all types of leaves except Kelloggii, which would actually be consistent with our thought when considering features of different leaf classes (figs. 12-14), where we said that the colour among leaves of this class differes quite a lot actually.

For the texture feature we see that it's extremely good for Kelloggii and Garryana which is actually surprising as Kelloggii leaves looked very generic texture wise, and I thought that they had no texture features which would make them suitable for this method. Garryana on the other hand has those very small wrinkles, which I assume allowed it to have such a high score when using texture feature.

When using all features, for every leaf it works with pretty much 1.0 score, except for Kelloggii, which is 5 percentage points short of 1.0. Interestingly it got lower score when using all features than when using just Local Binary Patterns. Which, I assume, means that texture is it's most prominent feature.

5. Conclusions

The most obvious conclusion is that, using shape for our dataset was a bad idea. It yielded by far the worst results. Texture was much better reaching over 80% accuracy. On top of all features we have histogram, contrary to my initial thoughts. I find it interesting that colour turned out to be the best feature to use for classifying leaves by species.

When it comes to classifying leaves and using machine learning for that, the more regular and distinct each class is the better. Using all the features, namely colour, shape and texture we managed to obtain nearly 100% accuracy (98, 99% to be exact). Each method used was found to have it's own, usually unique weaknesses.

Well for the shape feature almost all of the leaves could be considered as weaknesses. In particular it struggled with leaves that looked similarly, so Circinatum and Glabrum, which is rather expected since it tries to categorize leaf by it's shape so if it finds one that is similar, then we might get errors.

Texture struggled the most with Glabrum leaves. In this case struggled is still much better than the best result of shape method. On the other hand it was the only method (local binary patterns to be exact) which managed to classify Kelloggii leaves with 100% accuracy.

For our data set, combining those 3 methods was enough to reach almost 100% accuracy, which was rather surprising considering the fact that we didn't have that big of datasets to train from and that some of the leaves looked rather irregular to human eye.