ML Project Documentation

Data preprocessing functions

Usable functions

`loading_artists_labels(path, artists, resize_scale)`

Loads all samples for all specified artists, resizes the images, and labels them.

Output: `data, Y`

where:

`data` – column vector of images as arrays with pixels

`Y` – a column vector of labels (integers starting at 0)

Input instructions:

`path` – path to a folder in which there are 50 folders (1 for each artist) with paintings

`artists` – list of indices of artists that we want to load, e.g. [24, 42, 48], which corresponds to Cezanne, Rembrandt, and van Gogh in our dataset

`resize_scale` - % of how much we want to resize our images

`indices(path_to_folder)`

Returns a pandas dataframe with indices that are assigned to artists. It allows to choose the artists, whom we want to load.

Input instructions:

`path_to_folder` – path to a folder in which there are 50 folders (1 for each artist) with paintings

`grayscale_blur(images)`

Returns a dataset transformed to grayscale and blurred.

Input instructions:

`images` – dataset of brg photos returned by a loading function

Support functions (likely not useful)

`read_artist(name, path_to_fold)` – reads images of 1 artist specified by name

`resize(images,scale_percent)` – resizes images within a given dataset by a given percent

`load_resize(name,scale, path_to_folder)` – loads and resizes images of 1 specified artist by name

`artist_no_paintings(index,path_to_fold)` – yields the number of paintings in the dataset by 1 artist

Training functions

Usable functions

`softmax(features,labels,folds_no=5)`

Performs a SoftMax regression classification on the given dataset. By default, the function performs learning folds_no=5 times.

Output: `avg_test_acc, std_test_acc`

Where:

`avg_test_acc` – average test set accuracy of learning folds_no times

`std_test_acc` – standard deviation of test set accuracies from folds-no fold learning

Input instructions:

`features` – dataset of features in the form of a $m\ x\ n$ matrix, where m is the number of samples, and n is the number of features

`labels` – numerical labels for the above dataset as a column vector

`folds_no` – (default parameter) number of trainings that the function performs over different training sets

`GNB(features,labels,folds_no=5)`

Performs a Gaussian Naïve Bayes classification on the given dataset. Overall, the same as above

`forest(features,labels,trees_no=11,folds_no=5)`

Performs a Random Forest classification on the given dataset. Overall, the same as above. (the function also one-hot-encodes labels.

The function allows to specify the number of trees we want to grow. By default, it is set to `trees_no=11`, but for more artists, it will have to be bigger.

`knn(features,labels,neighbors_no=5,folds_no=5)`

Performs a K-Nearest Neighbors classification on the given dataset. Overall, the same as above.

The function allows to specify the neighbors we consider. By default, it is set to `neighbors_no=5`.

SVM(features,labels,folds_no=5)

Performs a SVM classification on the given dataset. Overall, the same as above.

AdaBoost(features,labels,folds_no=5)

Performs an AdaBoost classification on the given dataset. Overall, the same as above.

method_list()

Return the list of methods available for use in the script.

Output: `list of methods`

Support function

`fold_training(method,feature,labels,folds_no=5)`

Every learning model in this script relies on this function to perform k-fold validation.

Training optimization functions

Usable functions

Generally speaking, functions that allow to fine-tune the learning methods that need parameter specification

`best_neighbors(features,labels,block, folds_no = 5)`

Finds the number of neighbors for which KNN yields the highest accuracy. The function essentially performs KNN classification on the given dataset for a set of given parameters (numbers of neighbors). By default, the function performs learning folds_no=5 times. The function also visualizes accuracies for tested numbers of neighbors.

Output: `max_index` & scatter plot with accuracies

Where:

`max_index` – the parameter which yielded the highest accuracy

Input instructions:

`features` – dataset of features in the form of a $m\ x\ n$ matrix, where m is the number of samples, and n is the number of features

`labels` – numerical labels for the above dataset as a column vector

`block` – the segment of numbers of neighbors for which the function will apply KNN and find accuracy, e.g. `[1,29]` will evaluate KNN for number of neighbors from 1 to 28

`folds_no` – (default parameter) number of trainings that the function performs over different training sets

`best_forest(features,labels,block, folds_no = 5)`

Essentially the same as the above function but applied to Random Forest and optimizing for the number of trees.

Integrating functions & visualization

Usable functions

`evaluation(features,labels,no_neighbors=3, no_trees= 21, num_folds=5,g_time = False)`

Performs learning on the given dataset for all training functions written, and then visualizes the accuracies yielded on a bar graph.

Output: `acc_pandas` & graph with accuracies

`acc_pandas` – a table of accuracies yielded by respective learning methods on a given dataset

Input instructions:

`features` – dataset of features in the form of a $m \; x \; n$ matrix, where m is the number of samples, and n is the number of features

`labels` – numerical labels for the above dataset as a column vector

default values:

`no_neighbors = 21` – number of neighbors for KNN

`no_trees= 21` – number of trees for random forest

`num_folds=5` – number of times the models are evaluated

`g_time = False` – a parameter to suppress output when using the graph_random function; feel free to ignore it

Adding a new learning method to be included in the evaluations:

add this line below the last "method_accuracy"

```
method_accuracy =
np.vstack((method_accuracy,['name_new',round(new_method_function(features,labels)[0],2
)]))
```

where:

`'name_new'` – string to be displayed as the name of the method

`new_method_function` – function that starts learning on a dataset with the new method; make sure the new function returns accuracy

`graph_random(numbers,path,resize_scale=40)`

Performs learning on for given numbers of artists that are randomly chosen artists. It utilizes the `evaluation` function, and so learning is done for methods implemented there (currently SoftMax, Random Forest, Gaussian Naïve Bayes, KNN).

Output: `none`

The only output is a bar graph with accuracies for different numbers of artists, over different learning methods.

Input instructions:

`numbers` – a list of numbers of artists we want to evaluate; for example, passing [2, 3, 7] will implement learning models to 2 randomly chosen artists, then 3 randomly chosen artists and then 7 randomly chosen artists

`path` – path to a folder in which there are 50 folders (1 for each artist) with paintings

`resize_scale` - % of how much we want to resize our images

Feature extraction

Usable functions

`histogram_feature(images,bins_no=8,range=256)`

Extracts histograms of a gives set of images in RGB and returns them as a dataset ready for learning. Note: the function can be adjusted for other types of data, such as images in grayscale or HSV

Output: `histogram_dataset`

`histogram_dataset` – featurized samples as a matrix of size $m\ x\ (3 * bins\_no)$, where m is the number of samples and $(3 * bins\_no)$ the number of features.

Input instructions:

`images` – a dataset with images that are to be featurized

`bins_no=8` – the number of bins in every channel, in which pixel distribution will be shown; set to 8 by default (I usually had the best answers with 8)

`range=256` – the range of values in the dataset; for our images in RBG, it is from 0 to 256.

`kmeans_colors(images,no_colors, returned_colors=3)`

Extracts a few colors to which images form a dataset can be shrunk using k-means. Note: it is very computationally costly but might offer decent accuracy for higher no_colors.

Output: `top_colors_average`

`top_colors_average` – the average RGB values for the `(returned_colors)` colors that have the highest share of pixels clustered to them with k-means. Output has the form of a $m\ x\ n$ matrix, where m is the number of samples, and n is the number of `returned_colors`

Input instructions:

`images` – a dataset with images that are to be featurized

`no-colors` – the number of colors to which we want to shrink the images

`returned_colors=3` – the number of colors with the highest share we want it to return (set to 3 by default as using more didn't yield any gains)

hsv_histogram_feature(images,bins_no = 8)

Same as histogram_feature but with HSV color scheme.

Output: `histogram_data`

Input instructions:

`images` – a dataset with images that are to be featurized

`bins_no = 8` – the number of bins in every channel of HSV; set to 8 by default

hu_feature(images)

Yields Hu moments of an image – different ways of quantifying shapes.

Output: `hu_data`

Input instructions:

`images` – a dataset with images that are to be featurized

haralick_feature(images)

Yields Haralick textures of an image – quantifying pixel intensity consistencies

Output: `haralick_data`

Input instructions:

`images` – a dataset with images that are to be featurized

canny_feature(images, t1 = 1, t2 = 11)

Applies a Canny Edge detection feature to grayscale images (grayscale computed inside), sums all the values and normalizes these sums. It intends to quantify how many edges there are

Output: `sums`

`sums` – a vector of sums described above

Input instructions:

`images` – a dataset with images that are to be featurized

`t1` – threshold 1 ??; set to 1 by default

`t2` – threshold 2 ??; set to 2 by default (I was able to get highest accuracies with these default settings)

HOG_feature(images)

Yields sums of the Histograms of gradients for an image.

Output: `HOG_data`

Input instructions:

`images` – a dataset with images that are to be featurized

SIFT_no_feature(images)

Applies a Scale-Invariant Feature Transform (SIFT) algorithm to grayscale images, and finds the keypoints of that image. Calculates the number of keypoints for each image.

Output: `kp_no`

`kp_no` – a vector of the number of keypoints

Input instructions:

`images` – a dataset with images that are to be featurized

LBP_feature(images,numPoints=36,radius=15)

Finds consistency in an image on a local scale with Local Binary Patterns.

Output: `hist_store`

`hist_store` – a histogram of LBPs, given the preset number of points and their radius

Input instructions:

`images` – a dataset with images that are to be featurized; numPoints, radius – default parameters of local points

Harris_Corner_feature(images,bs=2,ks=3,k=0.04)

Extracts Harris Corners – points with local neighborhoods of 2 dominants edge directions.

Output: `corners_storage`

`corners_storage` – a vector of the location of corners

Input instructions:

`images` – a dataset with images that are to be featurized; bs, ks, k – corner detection params

Feature integration & simulations

Usable functions

`feature_list(names=True,indices=True)`

Returns a pandas dataframe of features and their respective indices, with which we can featurize our dataset.

Output: `features`

Input instructions:

`names` – `ignore!` if False, it returns the actual featurizing functions (used in other functions)

`indices` – if False, it returns a list of features instead of a pandas dataframe (no indices here)

NOTE: If you hope to add a new feature and include it in all the useful functions below, you do so in this function. Add this line right below the last analogous line.

`features = np.append(features,` **`EXACT_NAME_OF_THE_FUNCTION`**`)`

`featurizing_all(images)`

A function that applies all available features to a set of given images. Returns an array (#images x #features).

Output: `featurized_data`

Input instructions:

`images` – a dataset with images that are to be featurized

`featurizing_sel(images, features, names = False)`

A function that applies selected features to a set of given images. Returns an array (#images x #features).

Output: `featurized_data`

Input instructions:

`images` – a dataset with images that are to be featurized

`features` – a list of indices that correspond to indices of available features; for example, according to the list of features below, passing [0, 2, 5] will apply the histogram feature, hu feature, and the canny feature to the given set of images

index	features
0	histogram_feature
1	hsv_histogram_feature
2	hu_feature
3	haralick_feature
4	HOG_feature
5	canny_feature
6	keypoint_no_feature

`all_feats_together(images, labels)`

A function that applies all available features to a dataset, trains all available models on these features together, and visualizes performances.

Output: *`displays a bar graph and numerical performances`*

Input instructions:

`images` – a dataset with images that are to be featurized

`labels` – numerical labels for the above dataset as a column vector

`all_feats_fit(images, Y)`

A function that applies all available features to a dataset, trains all available models on these features separately (note the crucial difference as compared to all_feats_together), and visualizes performances of different features and methods.

Output: *`a pandas dataframe of performances (+ displays a bar graph)`*

Input instructions:

`images` – a dataset with images that are to be featurized

`Y` – numerical labels for the above dataset as a column vector

`feats_fit(images, Y, list_of_combinations)`

A function that applies selected combinations of features to a dataset, trains all available models on these combinations of features, and visualizes performances of different features and methods. It allows to train a dataset on different sets of features and compare their performance.

Output: *`a pandas dataframe of performances (+ displays a bar graph)`*

Input instructions:

`images` – a dataset with images that are to be featurized

`Y` – numerical labels for the above dataset as a column vector

`list_of_combinations` – (similar to featurizing_sel); a list of lists of indices that correspond to features we want to combine and train on. For example, according to the list of available features, if we hope to train the models on hu_features and SIFT_no_features together, and also (in a different simulation) haralick features and HOG features together, we will pass:

`[[2, 6], [3, 4]]` to the function.

index	features
0	histogram_feature
1	hsv_histogram_feature
2	hu_feature
3	haralick_feature
4	HOG_feature
5	canny_feature
6	keypoint_no_feature