Introduction to Computational Creativity, 2015
Department of Computer Science, University of Helsinki

We again include some optional tasks. You are encouraged to work on them, but this is not required.

Assignment 4: Creativity as search; Image Processing and Machine Learning

This assignment is due by Wed 23 Sep at 23:59 am.

This week we work on two topics: (1) formalization of creativity as search, by Wiggins, and (2) image processing and machine learning. Within the latter topic, we are going to classify small images of faces with image processing and machine learning techniques. These exercises build up for the next week, when we are going to use our learned model as an evaluator in a CC task.

1. Read this paper by Geraint A. Wiggins: A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems* 19 (7): 449–458, 2006. (pdf available at least from the university network).

2. * Consider Markov chains as implemented in the very first assignments, and their use in generation of new text. Outline in your diary how such a system could be described using Wiggins' concepts. What could be U, R, E, T in such a Markov chain? Also give brief arguments why. Note: There is no single correct answer -- there can be different arguments for different descriptions. (Also remember that Wiggins' model does not need to match one-to-one with the architecture.)

3. * Now consider your program for producing limericks, in the last tasks of the previous assignment. (Alternatively, if you didn't implement a limerick generator, think of a system you could have built. Describe it briefly.) Describe the main points of that program using Wiggins' framework.

4. * Is the limerick system above transformationally creative, i.e., does it involve search in the meta-level, i.e., does it modify R, E, or T in any sense? Can you think of ways of making in transformationally creative?

5. Install the required Python libraries. In this week's tasks, we need the following Python libraries: scikit-learn, and scikit-image. Both have NumPy (>= 1.6.1), and SciPy (>= 0.9) as requirements. All the packages are bundled in Anaconda, which probably is the easiest way to get them if you have not used Python before the course (or are running Windows). (Please note: Anaconda comes with its own Python interpreter and `ipython`, so you have to know which Python to call in order to have the libraries in your Python environment.) For installing Anaconda, see its own documentation. In order to install the libraries with pip, we have to install NumPy before other libraries due to certain limitations in the pip's installation procedure (you might need sudo):

```
$> pip install numpy
$> pip install scipy
$> pip install scikit-learn
$> pip install scikit-image
```

To check that you have the required libraries installed to your favourite (i)python environment, make sure that the following commands don't raise any errors:

```
$> ipython
-- in Python interpreter
In [1]: import numpy
In [2]: import scipy
In [3]: import sklearn
In [4]: import skimage
```

About NumPy: NumPy is the core library for scientific computation in Python. It has been implemented in C with Python bindings to get C's speed (with very little overhead) and Python's convenient syntax. NumPy has a fundamental datastructure called Numpy array (ndarray). In many ways NumPy arrays behave like Python's native lists, but there are some limitations, e.g. they have fixed size and a data type. You can create Numpy array from a Python list with:

```
In [1]: import numpy as np # here we give alias np to numpy for shorter
notation
In [2]: a = [[1,2],[3,4]] # make normal python list
In [3]: ar = np.array(a, dtype=np.uint8) # convert the list to a numpy array
with unsigned byte as a data type
In [4]: ar
```

```
Out[4]:
array([[1, 2],
       [3, 4]], dtype=uint8)
In [5]: ar.shape # array's size
Out[5]: (2, 2)
In [6]: b = [[1,2],['a','b']]
In [7]: np.array(b, dtype=np.uint8) # raises error because of data type
restrictions
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-cdcb55d6c9b5> in <module>()
----> 1 np.array(b, dtype=np.uint8)

ValueError: invalid literal for long() with base 10: 'a'
```

More instructions for NumPy can be found for example in http://www.python-course.eu/numpy.php We will also go though some basic NumPy operations in the lab session on Monday 21.9..

6. Classification as a supervised machine learning task.
    a. If you have no previous background in machine learning (or need to refresh your memory), familiarize yourself with the (binary) classification task. Some basic material:
        i. https://en.wikipedia.org/wiki/Binary_classification
        ii. https://en.wikipedia.org/wiki/Statistical_classification
    b. * Briefly write in your learning journal what (binary) classification means, how it is (in its simplest form) executed, and how it can be evaluated. 5-10 sentences is enough.
    c. * Briefly write in your learning journal any thoughts about how could a previously learned classifier (model) be used in CC?

In the following tasks, we will walk through the steps of building a classifier: reading the data, extracting features from it, training a classifier, and testing it. For those of you with background in machine learning some of the detailed steps and instructions are unnecessary, but we hope they are useful for those with less previous experience with machine learning.

7. *Reading the dataset.*

a. Download and extract the following face image dataset. Examine its folder structure. http://cbcl.mit.edu/software-datasets/FaceData2.html
b. Using `skimage.io`, write a function `read_images(folder, ext)` which reads all the images (as gray scale) with the given file extension `ext` from a folder and returns them in a list. HINT: use `os.listdir()`
c. Write a function `read_samples(pos_folder, neg_folder, ext=".pgm")` which reads all the images with the extension from `pos_folder` and `neg_folder` and returns both of them (in a predetermined order). HINT: Python functions can return multiple objects, just write comma between the attributes, e.g. `return a,b`

8. *Extracting features from the images.* For each data point (in our case an image), we need to extract some features from it for the classifier to find the similarities inside, and dissimilarities between, the classes.

   Write a function `extract_features(image, types=['pixels'])`, which takes as an input **gray scale** image (as a NumPy array) and a list, and returns a vector of features as a NumPy array. Each item in the `types` is a string which corresponds to certain type of features to be extracted from the image. For each type in `types`, the correct features are extracted from the image, and the feature vectors are concatenated to form the returned vector. For now, we only consider one type of features:
   a. `'pixels'`: flatten the input image to a vector and use it as a feature vector, i.e., each pixel is a feature.

9. *Training the classifier.*

   Write a function `train(pos_folder, neg_folder, ext='.pgm', feature_types=['pixels'])` that:
   a. for `pos_folder` and `neg_folder`: extracts a feature vector (using `extract_features`) for each image in the folder and creates and NumPy array of all the feature vectors.
   b. creates label vectors for positive and negative samples. Positive samples are labeled as 1 and negative samples are labeled as 0.
   c. concatenates the feature vector arrays for positive and negative samples. HINT: use `numpy.concatenate`
   d. concatenates the label vectors for positive and negative samples

e. randomizes the order of samples and uses the same randomization indices to shuffle the concatenated label vector. HINT: use, e.g. `numpy.random.permutation`
f. calls `.fit(X, y)` function for a binary classifier of your choice, e.g. `AdaBoostClassifier`, or `SVC`. Optional: You can also make the `train`-function to accept as a parameter a classifier instance you would like to fit.
g. saves the resulting classifier as a pickle (use joblib). Optional: You can also add the option of saving the classifier (and its filename) as a parameter. HINT: It is a good idea to include the classifier's class name, and the feature types included in the training, in the pickle's name.

Train the classifier by calling the `train`-function with `pos_folder='train/face/'` and `neg_folder='train/non-face/'`

10. *Evaluating the classifier with a test set.* After training, we of course need to evaluate how well our classifier works. For this purpose we are going to write a test function that (1) uses our trained classifier to predict the labels for the test set and (2) outputs some meaningful statistics from the predictions.

    Write a function `test(pos_folder, neg_folder, ext='.pgm', feature_types=['pixels'], pickle='classifier.pkl')`, that:
    a. extracts a feature vector (using `extract_features`) for each image in both folders
    b. loads the saved classifier `clf` from a given pickle
    c. calls `clf.predict()` for positive samples and negative samples.
    d. prints out the percent of correct classifications for positive and negative samples, separately.
    e. (Optional) output more statistics for the evaluation.
    f. * Write in your learning journal the classification results (percents of correct classifications) given by the `test`-function, when `pos_folder='test/face/'` and `neg_folder='test/non-face/'`. Are the pixel values good features for image classification?

11. More features! The performance of a classifier depends, among other things, on the features used and how informative they are for the problem at hand. We'll now consider more complex features than just individual pixels.
    a. Local Binary Patterns
        i. * Familiarize yourself with local binary patterns (LBP). Write a short description.
        ii. Write a function `extract_lbp` that calculates local binary patterns (for now, you can consider only P=8, R=1) from an image, flattens the result to a vector and returns it. You can use `skimage.feature.local_binary_pattern` to extract LBPs.
        iii. Add the `extract_lbp` as an option to `extract_features`-function's `types`-parameter. (You can name the option differently, e.g. as `'lbp'`).
        iv. * Train your classifier with LBP-features and evaluate its classification results. Are LBPs better than single pixels as features? Can you think of a reason for their performance?
    b. Regional LBP histograms
        i. write a function `extract_lbp_histogram(image, regions=(3,3))`, that calculates the LBPs from an image, splits image to NxN regions (that is, with `regions=(3,3)` image of size 30x30 would be divided into nine 10x10 pixel regions), and calculates histogram of the possible values for each regions. Then, it concatenates the histograms to a single NumPy array, and returns it. HINT: use `numpy.array_split`
        ii. Similarly as in 9.a., add the `extract_lbp_histogram` as an option to `extract_features`-function's `types`-parameter. (You can name the option differently for different parameters, e.g. as `'lbp_histogram_3x3', lbp_histogram_5x3,` etc.)
        iii. * Train your classifier with regional LBP histograms as features and evaluate its classification results. Are they better than basic LBPs or pixels as features?
    c. (Optional) Histogram of Oriented Gradients
        i. Familiarize yourself with histogram of oriented gradients (hog).
        ii. write a function extract_hog(image), that calculates hogs from the image using `skimage.feature.hog` and returns it.
        iii. Similarly as above, add the `extract_hog` as an option to the `types`-parameter of the `extract_features`-function.

iv. * As above, train your classifier with hogs, and report your results with the test set to your learning journal.

d. *** (Optional, but very strongly recommended esp. for those who have taken courses in machine learning)** Investigate the vast plane of image processing methods and come up with your own features to use in the classifier training. You can also try different combinations of the above features. Which ones give you the best results? Write your results in the learning journal.

12. * Briefly write in your learning journal the thoughts inspired by the classification and image processing tasks. **NB**: You can do this task, even if you did not manage to do all the previous exercises.

IMPORTANT: Once you are done with the tasks and the learning journal, remember to save a version of it in Overleaf, and then send the pointer and version identifier in an email to slinkola@cs.helsinki.fi.