Image Classification using Informative Features

Goal:

The goal of the project is to extract informative fragments from training data, for the purpose of classification of new data. You are asked to implement the ideas that were presented during the Recognition lectures. Detailed explanation on how to do this is below.

Data:

You are provided with two different datasets "cars" and "faces". Your code should be executed twice, once for each of the sets.

Non-class images:

Training   Testing

Class images:

I. "Cars" dataset: Training   Testing

II. "Faces" dataset: Training Testing

Training stage:

1. Data preparation: for each class prepare two data structures (matlab struct) – "training" and "testing" both with two fields:

- "image_path" – cell array of full paths to the images (length N)
- "image_labels" – binary vector of image labels (length N), 1 for class, 0 for non-class.

2. Fragment extraction from the class images: write a function called "FragmentExtraction.m" that performs the following

- Input: the struct with the data of the class ("cars" or "faces")
- Reads all the class images (class training data)
- For each of the images you should divide the image to overlapping (overlap of half the fragment size) rectangular fragments. Fragment sizes: 30x30, 40x40 (first divide the image to 30x30 fragments, then to 40x40 and gather all the fragments of different sizes together).

- Output: Cell array of fragments, the total number of fragments you will get is: 1 x (M * N) (where M is the number of fragments per image, and N is the number of images in the training set)
- **The function prototype:**

  fragments = FragmentExtraction(training);

2. Calculate fragment-image similarity measure:

- You can use the function fast_NCC_NoMex.m to calculate the maximum Normalized Cross Correlation (NCC) between each of the input images and each of the fragments.
- This should be done for each of the class and non-class images. Build a matrix R of size N x M, entry R(i,j) contains the maximal NCC between fragment #j and image #i.
- Compute the optimal threshold T for each fragment as explained here (you should try different thresholds and decide what is the best one according to the highest Mutual Information (MI), one way to try thresholds is to take all the values from the NCC vectors). Threshold the maximal NCC vectors (columns of R) to get binary features, you should get one threshold per fragment.
- **Output**:
  C, NC - two matrices one for class one for non-class images. Each matrix is of size N x M , each entry is 0/1 according to the thresholded maximal NCC.
  thresholds - thresholds chosen during the computation (vector of length M).
  MI - vector of mutual information of each fragment with the class corresponding to the chosen threshold (vector of length M).
- **The function prototype:**

  [C, NC, thresholds, MI] = ThresholdLearning(training, fragments);

3. Feature selection (choose the best fragments)

- Write a function called "FeatureSelection.m" to select the best fragments to use in the classifier.
- Choose the best 50 fragments.

  Use the max-min algorithm: you choose the first fragment as the one with the maximal MI and all subsequent fragments as:

$$\max_i \min_j MI(F_i; C|F_j)$$

  where *i* goes over the remaining fragments and *j* goes over the already chosen fragments. The conditional mutual information is defined as:

$$MI(Fi ; C \mid Fj) = H(C \mid Fj) - H(C \mid Fi, Fj)$$

with conditional entropy *H(C | Fj)* is defined as in the threshold calculation, and $H(C \mid F_i, F_j) = -\sum_{C,F_i,F_j} P(C, F_i, F_j) \log P(C \mid F_i, F_j)$ where *P(C,Fi,Fj)* is computed empirically by going over all possible triplets of binary values for *C, Fi* and *Fj*. In fact, the value of “C” chooses between the C and NC matrices and indices *i* and *j* choose two columns in them, all that remains is to count the relevant “events” and divide by *N*.

- **Output**: frag_idx - are the chosen fragments indices
- **The function prototype**:

  frag_idx = FeatureSelection(C, NC);

4. Compute feature weights

- Use the Naïve Bayes model:

  Naïve Bayes model: To classify an image we would like to compute the log-likelihood ratio test of the form:

  $$\mathrm{LLRT} = \log \frac{\mathrm{P(F1,\ F2,...,\ Fn} \mid \mathrm{C}=1)}{\mathrm{P(F1,\ F2,...,\ Fn} \mid \mathrm{C}=0)} > \mathrm{Threshold}$$

  where the “Threshold” stands for the “real-life” ratio between class and non-class, and *Fi* are the feature values (binary in our case). According to the NB model:

  $$\mathrm{P(F1,...,\ Fn} \mid \mathrm{C}) = \prod_{\mathrm{i}} \mathrm{P(Fi} \mid \mathrm{C})$$

  Assign the feature weight to be:
  $W_i(a) = \log P(F_i = a \mid C = 1) - \log P(F_i = a \mid C = 0)$
  for $a \in \{0,1\}$ and $P(F_i = a \mid C)$ computed empirically from the training set. We can then compute the log-likelihood ratio test for a given image as:

  $$LLRT = \sum_i W_i(F_i)$$

  where *Fi* is a binary indicator of whether feature #i was or was not detected (surpassed its threshold) in the given image.
- **The function prototype**:
  W = ComputeWeights(C, NC, frag_idx); (*W* should be of length 50 = length of *frag_idx*)

Classification -- Testing stage:

Build Naïve Bayes classifier that takes the best fragments from the training stage and classify the test data to class and non-class images. Specifically,

- Write a function "classify.m" which does the following:
- For each test image use the NCC function to get max NCC for each fragment from the chosen set: put them into vector X.
- Apply the same threshold as in training step (says whether particular fragment is detected in the image or not) to get a binary vector Y.
- Compute the image score as: $\text{Image Score} = \sum_i W_i(Y_i)$ using the weights computed by the *ComputeWeights* function.
- Decision whether the image is class or non-class should be based on that score.
- **The function prototype:**

  Scores = classify(testing, fragments, frag_idx, W);

- You should use the ROC curve to assess the performance of your classifier (sample code). You can try changing the various parameters of the training such as fragment sizes, number of chosen fragments, feature selection algorithm, etc. and see how it affects the ROC. One way to choose the better ROC is by looking on the area underneath it (the more the better).

Good Luck!