Gender Recognition

Cihan SARI

January 11, 2013

1 Introduction

One of the most basic classification regarding humans is gender. It is definitely the very first thing a person decides on sight of another person. This project aims to make decision of the gender by using facial attributes in computer vision. A human can easily make this decision from faces (above 95% accuracy [5]). Yet, it is still a challenging task for computer vision. However, this is mostly because such attribute classification problems have not been worked as much as more popular problem if individual face recognition[2]. This project is focused on gender recognition in face images and videos using computer vision techniques.

2 Previous Work

Facial characteristics are probably the most common features for humans to identify gender[11]. The face region, which may include external features such as the hair and neck region, is used to make gender recognition[2]. Therefore, many authors have used face image to distinguish gender of the subject[8][13][12] (and many many others that are not cited). Gutta used mixture of experts on gender and ethnic classification fo human faces, and pose classification on FERET database of facial images[8]. Moghaddam used nonlinear support vector machines for appearance-based gender classification with low-resolution faces processed from 1,755 images from FERET face database[13]. Zhen Li used spatial Gaussian mixture models (SGMM) which inherits all the merits of GMM, such as precise appearance description and robustness to image misalignment on the yamaha gender and age (YGA) database[12].

3 Methodology

3.1 Face Dataset

FERET (Face Recognition Technology) is a widely used open dataset for evaluation of face recognition algorithms, and has also been used by many researchers for face gender recognition[2]. It contains 4,215 frontal face images from 2,712 subjects.[14]. 2,720 face images, from which 1715 belong to male and 1005 belong to female subjects from FERET dataset are used as training data in this project.

Vistek face dataset is collected from co-workers with a webcam. It contains 70 male and 34 female images, up to 10 images from the same person. This dataset is used as first test set. LFW (Labeled Faces in the Wild) is a very rich database of face photographs designed for studying the problem of unconstrained face recognition[10]. It contains over 13,000 images of faces collected from the web. 10,103 male and 2,927 female images from this dataset is used to calculate second test performances. Many of the LFW images are from arbitrary poses. Therefore, a filtered set from LFW is being created for more meaningful test results. Currently, 145 male and 149 female images are filtered to be used as third test set.

3.2 Preprocessing and Feature Extraction

Some preprocessing is necessary to only teach the classifier important features. Too dark, over-exposed images or different image sizes will be learnt by the classifier such as SVM to produce artificially better performance[7]. Therefore face images are,

- Viola&Jones OpenCV implementation to detect faces,
- Reduced 22% from top and 16% from left and right regions,
- Smoothed using a gaussian blur with size \( \frac{\text{height}}{12} \),
- Downsized to 64 \( \times \) 64 to reduce number of pixels and also size-align images.
• Contrast normalized using histogram equalization,
• Concatenated DCT and LBP vectors are used as feature vectors

3.3 Classification

Support vector machines (SVMs) are used to classify gender from feature vectors. There are some key advantages of random decision forests (RDF) over SVM. Therefore, aim of this project is to increase RDF’s performance to match (or hopefully surpass) SVM’s performance.

3.3.1 SVM

Support vector machines (SVMs) are trained using the 832 dimensional feature vector, per face image. Following steps are used for classification.

• Five-fold cross validation is used on the training set (FERET) to train SVM classifier.
• Best average performing classifier is used on the testing sets to determine testing performance.

Training and testing set performances are reported as seen in figure 1.

```
<table>
<thead>
<tr>
<th></th>
<th>Female</th>
<th>Male</th>
<th>Mean Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vistek</td>
<td>94.10%</td>
<td>97.10%</td>
<td>96.20%</td>
</tr>
<tr>
<td>LFW(F)</td>
<td>91.30%</td>
<td>76.00%</td>
<td>84.00%</td>
</tr>
<tr>
<td>LFW</td>
<td>84.30%</td>
<td>76.20%</td>
<td>78.00%</td>
</tr>
</tbody>
</table>
```

Figure 1: Test performances from SVM

4 Random Decision Forests

Recent classification problems, i.e. face recognition, gender recognition, medical diagnosis and document retrieval, often have many input variables, in the hundreds or thousands, with each one containing only a small amount of information. Although a single tree classifier will have relatively poor performance on these cases, combining trees grown using random features can produce improved accuracy[4].

Randomized decision forest (RDF) has several key advantages:

• Fast training, even faster testing[3],
• Conceptually simple algorithms[6],
• Built-in feature selection mechanism[6],
• Can cope with missing features[6],
• Can cope with all types of features: nominal, ordinal or numerical[9],
• Inherently multi-class (SVM and AdaBoost require sub-optimal work around)[6].
4.1 Forests from Leaves

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest[4]. Each tree consists of a set of decision nodes, where the training data that reaches to node is split into deeper nodes or leaves. This process continues, until all the nodes in all the trees lead to a leaf. Each leaf simply represents the class which majority of the training instances that falls into the leaf belongs.

As previously mentioned, each node in each tree represents a split. Searching such a split point that maximizes purity of the children nodes is computationally very expensive on large datasets with a large amount of feature dimensions. To reduce the complexity, only a small random subset of the features are used to compute purity of the children for each node. Such a reduction not only heavily decreases computation time, but also makes each tree robust with respect to noise[4].

4.2 Parameters

There are several parameters that has an effect on the structure of the rdf. On this project,

- Gini’s index is used for splitting criterion.
- All the trees are trained with same instances of the data.
- All the trees are trained with same feature dimensions.

Therefore, each tree that forms the forest is trained with very same parameters. However, variety is achieved by bootstrap aggregating[1] at each node. Only a limited number of features are computed to measure purity. Feature count and other parameters are measured against validation performance as a grid search to optimize parameters.

- Forest size (M) is the number of trees that are used to generate the forest. ($M = 10, 50, 200$)
- (Minimum) Leaf size (L) is the minimum number of instances that should belong to each leaf. ($L = 200, 100, 10$)
- Feature count (K) is the number of features that are randomly picked from the feature space to measure the splitting criterion for each node. ($K = 10, 40, 100$)

As can be seen in figure 2, each node consists of a decision where only one feature is thresholded. Eventually, each path in the tree leads to a leaf, where the classification occurs. Random decision forest consists of M number of such trees, where in every node K number of features are considered for split. If there is purity inside the node, or if there are less than L number of instances remaining in the node, that node is split no further, and becomes a leaf which represents the class which majority of its instances belong to.

Figure 2: A sample tree from the forest
4.3 Validation Performance

Feret dataset is split into two groups for training and validation. 750 male and 750 female instances are used for training, and the remaining 965 male and 255 female instances are used to measure validation performance. Validation performance is computed for each combination of the M,K and L. All 27 different cases and their respective forest properties are given in figure 3. In this table:

<table>
<thead>
<tr>
<th>Forest Size</th>
<th>Leaf Size</th>
<th>Features to calculate</th>
<th>Female Perf</th>
<th>Male Perf</th>
<th>Avg Perf</th>
<th>Flat Avg Perf</th>
<th>Features Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>20</td>
<td>200</td>
<td>10</td>
<td>83.53%</td>
<td>82.38%</td>
<td>82.62%</td>
<td>82.96%</td>
</tr>
<tr>
<td>Case 2</td>
<td>20</td>
<td>100</td>
<td>10</td>
<td>80.39%</td>
<td>83.63%</td>
<td>82.95%</td>
<td>82.01%</td>
</tr>
<tr>
<td>Case 3</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>79.61%</td>
<td>86.53%</td>
<td>85.08%</td>
<td>83.07%</td>
</tr>
<tr>
<td>Case 4</td>
<td>50</td>
<td>200</td>
<td>10</td>
<td>81.18%</td>
<td>88.81%</td>
<td>87.21%</td>
<td>84.99%</td>
</tr>
<tr>
<td>Case 5</td>
<td>50</td>
<td>100</td>
<td>10</td>
<td>79.61%</td>
<td>91.61%</td>
<td>89.10%</td>
<td>85.61%</td>
</tr>
<tr>
<td>Case 6</td>
<td>50</td>
<td>10</td>
<td>10</td>
<td>79.22%</td>
<td>92.93%</td>
<td>89.59%</td>
<td>85.77%</td>
</tr>
<tr>
<td>Case 7</td>
<td>200</td>
<td>200</td>
<td>10</td>
<td>83.53%</td>
<td>89.12%</td>
<td>87.95%</td>
<td>86.32%</td>
</tr>
<tr>
<td>Case 8</td>
<td>200</td>
<td>100</td>
<td>10</td>
<td>80.00%</td>
<td>93.47%</td>
<td>90.66%</td>
<td>86.74%</td>
</tr>
<tr>
<td>Case 9</td>
<td>200</td>
<td>10</td>
<td>10</td>
<td>83.92%</td>
<td>92.95%</td>
<td>91.07%</td>
<td>88.44%</td>
</tr>
<tr>
<td>Case 10</td>
<td>20</td>
<td>200</td>
<td>40</td>
<td>85.88%</td>
<td>84.56%</td>
<td>84.84%</td>
<td>85.22%</td>
</tr>
<tr>
<td>Case 11</td>
<td>20</td>
<td>100</td>
<td>40</td>
<td>78.43%</td>
<td>85.39%</td>
<td>83.93%</td>
<td>81.91%</td>
</tr>
<tr>
<td>Case 12</td>
<td>20</td>
<td>10</td>
<td>40</td>
<td>81.57%</td>
<td>88.70%</td>
<td>87.21%</td>
<td>85.14%</td>
</tr>
<tr>
<td>Case 13</td>
<td>50</td>
<td>200</td>
<td>40</td>
<td>74.12%</td>
<td>90.57%</td>
<td>87.13%</td>
<td>82.34%</td>
</tr>
<tr>
<td>Case 14</td>
<td>50</td>
<td>100</td>
<td>40</td>
<td>76.08%</td>
<td>91.61%</td>
<td>88.36%</td>
<td>83.84%</td>
</tr>
<tr>
<td>Case 15</td>
<td>50</td>
<td>10</td>
<td>40</td>
<td>79.61%</td>
<td>92.54%</td>
<td>89.84%</td>
<td>86.07%</td>
</tr>
<tr>
<td>Case 16</td>
<td>200</td>
<td>200</td>
<td>40</td>
<td>78.82%</td>
<td>90.57%</td>
<td>88.11%</td>
<td>84.70%</td>
</tr>
<tr>
<td>Case 17</td>
<td>200</td>
<td>100</td>
<td>40</td>
<td>77.65%</td>
<td>92.02%</td>
<td>89.02%</td>
<td>84.83%</td>
</tr>
<tr>
<td>Case 18</td>
<td>200</td>
<td>10</td>
<td>40</td>
<td>78.82%</td>
<td>93.26%</td>
<td>90.25%</td>
<td>86.04%</td>
</tr>
<tr>
<td>Case 19</td>
<td>20</td>
<td>200</td>
<td>100</td>
<td>80.00%</td>
<td>75.34%</td>
<td>76.31%</td>
<td>77.67%</td>
</tr>
<tr>
<td>Case 20</td>
<td>20</td>
<td>100</td>
<td>100</td>
<td>78.43%</td>
<td>88.19%</td>
<td>86.15%</td>
<td>83.31%</td>
</tr>
<tr>
<td>Case 21</td>
<td>20</td>
<td>10</td>
<td>100</td>
<td>84.31%</td>
<td>88.19%</td>
<td>87.38%</td>
<td>86.25%</td>
</tr>
<tr>
<td>Case 22</td>
<td>50</td>
<td>200</td>
<td>100</td>
<td>77.25%</td>
<td>83.63%</td>
<td>82.30%</td>
<td>80.44%</td>
</tr>
<tr>
<td>Case 23</td>
<td>50</td>
<td>100</td>
<td>100</td>
<td>73.73%</td>
<td>89.33%</td>
<td>86.07%</td>
<td>81.53%</td>
</tr>
<tr>
<td>Case 24</td>
<td>50</td>
<td>10</td>
<td>100</td>
<td>80.78%</td>
<td>91.92%</td>
<td>89.59%</td>
<td>86.35%</td>
</tr>
<tr>
<td>Case 25</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>76.86%</td>
<td>85.70%</td>
<td>83.85%</td>
<td>81.28%</td>
</tr>
<tr>
<td>Case 26</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>75.69%</td>
<td>90.05%</td>
<td>87.05%</td>
<td>82.87%</td>
</tr>
<tr>
<td>Case 27</td>
<td>200</td>
<td>10</td>
<td>100</td>
<td>78.04%</td>
<td>92.95%</td>
<td>89.84%</td>
<td>85.50%</td>
</tr>
</tbody>
</table>

Figure 3: Validation performances for parameters

- Forest size (M),
- Leaf size (L),
- Feature count (K),
- Female validation performance that is calculated in the validation set consisting of 255 female instances,
- Male validation performance that is calculated in the validation set consisting of 965 male instances,
- Average validation performance that is calculated as weighted average of female and male validation performances,
- Flat average performance which is calculated as flat average female and male validation performances,
- Features used which is the ratio of features used in the forest for decision making to the total number of features (832) are provided.

In figure 4, error is calculated as 1 - flat average validation performance, and complexity parameter is taken as percentage of the features that are used in the forest. It can be seen that validation error decreases at first, but gradually increases as more and more features are introduced to the forest. Exception to this is case 9 (M=200, K=10, L=10) which has a validation performance of 88.44%. Following this point, second minimum validation error is achieved at case 8 (M=200, K=10, L=100) which has 86.74% validation performance. Considering the graph and the case results, it can be said that picking a low K and higher M performs much better than using a high K and low M. This result indicates that the logic underlying random decision forests actually works. Having a large number of weak decision trees performs better against forests consisting of trees that are more complicated in structure, but less in number.
4.4 Test Performances

Using previously obtained parameters for case 8 (from figure 3) new rdf model is generated using the whole Feret dataset. This new model consists of 53.49% of the total of 832 features. Vistek, LFW(F) and LFW gender recognition performances for this model are computed as seen in figures 5.

![Figure 4: Validation error vs Complexity](image)

<table>
<thead>
<tr>
<th></th>
<th>Female</th>
<th>Male</th>
<th>Mean Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vistek</td>
<td>82.35%</td>
<td>87.14%</td>
<td>85.58%</td>
</tr>
<tr>
<td>LFW(F)</td>
<td>77.85%</td>
<td>62.76%</td>
<td>70.41%</td>
</tr>
<tr>
<td>LFW</td>
<td>79.60%</td>
<td>62.09%</td>
<td>66.02%</td>
</tr>
</tbody>
</table>

![Figure 5: Test performances using 53.49% of the features with parameters (M=200, K=10, L=100)](image)

4.5 Feature Selection

RDF performs automatic feature selection due to its very nature - splitting until each leaf consists of desired purity. Therefore, it can be used to select features that best identify the difference between the classes. On huge datasets, rdf can be used as a data mining tool. Selected features could be used to train another classifier (such as SVM) which may be more suited for the classification.

To test the strength of the features selected by RDF, a new SVM model using only the features selected is trained. Test results obtained from this new SVM model can be seen in figure 6. Test performances of the SVM with approximately half the feature space is almost the same with the original SVM performances given in figure 1.

Feature selection is also beneficial if sources of the features are known. For example, on this project, each selected feature directly represents part of the face that is used to identify gender. This information could be used to determine parts of the face that change the most depending on the gender. Also, the same information could be used to unburden the feature extraction process by simply not extracting unnecessary features.
Figure 6: Test performances with SVM using rdf’s features which are 445 features out of total 832

5 Source Files

```matlab
clear; clc; close all;

% Settings
loadFeaturesFromFile = true;
loadRDFFromFile = true;

% Constants
timestamp = sprintf('%0.2i%0.2i%0.2i%0.2i%0.2i%0.2i', fix(clock));
rdfFile = 'rdf.mat';
featureFile = 'features.mat';
resultFile = ['results-' timestamp '.esp'];

% Backup
if (exist(featureFile, 'file') && ~loadFeaturesFromFile)
    mkdir(timestamp);
copyfile(featureFile, timestamp);
end
if (exist(rdfFile, 'file') && ~loadRDFFromFile)
    mkdir(timestamp);
copyfile(rdfFile, timestamp);
end

% Read features
if (~loadFeaturesFromFile)
    % Constants
    trainFiles = 'trainData_DCT_LBP_Feret.txt';
    testFiles = {testData_DCT_LBP_Vistek.txt};
    testFiles{end+1,:} = testData_DCT_LBP_LFW_subset.txt;
    testFiles{end+1,:} = testData_DCT_LBP_LFW.txt;
    testFiles{end+1,:} = testData_DCT_LBP_Feret.txt;
    % Read data
    [trainClasses, trainFeatures] = readData(trainFiles);
    [testClasses, testFeatures] = readData(testFiles);
    % Write data
    save(featureFile, 'testFeatures', 'trainFeatures', ...
         'testClasses', 'trainClasses');
else
    load(featureFile);
end

classes = unique(trainClasses);

% Train rdf
if (~loadRDFFromFile)
    % Pick training set
    idxTrain = true(2720,1);
    idxTrain(751:1005) = false;
    idxTrain(1005:751:end) = false;
    trainClassesT = trainClasses(idxTrain);
    trainFeaturesT = trainFeatures(idxTrain,:);
    validClasses = trainClasses(~idxTrain);
    validFeatures = trainFeatures(~idxTrain,:);
    nTreeList = [20, 50, 200]
    minLeafSizeList = [200, 100, 10]
    NVarToSampleList = [10, 40, 100];
    [nTreeMesh, minLeafSizeMesh, NVarToSampleMesh] = meshgrid(nTreeList, ..., minLeafSizeList, NVarToSampleList);
    perfMesh = zeros(size(nTreeMesh));
    class1PerfMesh = zeros(size(nTreeMesh));
```
class2PerfMesh = zeros(size(nTreeMesh));
featCountMesh = zeros(size(nTreeMesh));
avgPerfMesh = zeros(size(nTreeMesh));
errorRateMesh = zeros(size(nTreeMesh));

for idxIt = 1:numel(nTreeMesh)
    keyword = ['validModel-n'T' num2str(nTreeMesh(idxIt)) 'mL' ...
    num2str(minLeafSizeMesh(idxIt)) 'nV' ...
    num2str(NVarToSampleMesh(idxIt))];
    if (exist([keyword '.mat'], 'file'))
        rdfStruct = rdfTrain(trainFeaturesT, trainClassesT, ...
            'nTree', nTreeMesh(idxIt), 'minLeafSize', minLeafSizeMesh(idxIt), ...
            'NVarToSample', NVarToSampleMesh(idxIt));
        save(keyword, 'rdfStruct');
    else
        clear rdfStruct;
        load(keyword);
    end
    validClassesPredict = rdfPredict(rdfStruct, ...
        validFeatures, validClasses);
    classPerf = zeros(1, numel(classes));
    for idxClass = 1: numel(classes)
        class = classes(idxClass);
        idxValidClass = validClasses==class;
        validClassPredict = validClassesPredict(idxValidClass);
        perf = numel(find(validClassPredict==class))/numel(find(idxValidClass));
        classPerf(idxClass) = perf;
    end
    avgPerfMesh(idxIt) = mean(validClasses==validClassesPredict);
    perfMesh(idxIt) = mean(classPerf);
    class1PerfMesh(idxIt) = classPerf(1);
    class2PerfMesh(idxIt) = classPerf(2);
    featCountMesh(idxIt) = numel(rdfStruct.FeaturesSelected)/...
        size(validFeatures, 2);
    errorRateMesh(idxIt) = perfMesh(idxIt)*(1-featCountMesh(idxIt))/...
        size(validFeatures, 2));
    evalTxt = ['models.' keyword '= rdfStruct;'];
    eval(evalTxt);
end
save(rdfFile, 'perfMesh', 'avgPerfMesh', 'featCountMesh', 'NVarToSampleMesh', ...
    'class1PerfMesh', 'class2PerfMesh', 'minLeafSizeMesh', ...
    'nTreeMesh', 'errorRateMesh');
else
    load(rdfFile);
end

nTreeList = [20, 50, 200];
minLeafSizeList = [200, 100, 10];
NVarToSampleList = [10, 40, 100];
[nTreeMesh, minLeafSizeMesh, NVarToSampleMesh] = meshgrid(nTreeList, ...
    minLeafSizeList, NVarToSampleList);

% Headers
headerText = [...
    {'Forest Size'}, ...
    {'Leaf Size'}, ...
    {'Features to calculate'}, ...
    {'Female Perf'}, ...
    {'Male Perf'}, ...
    {'Avg Perf'}, ...
    {'Flat Avg Perf'}, ...
    {'Features Used'}, ...
    ];
nums = (1: numel(nTreeMesh));
caseText = [repmat('Case ', numel(nTreeMesh)), 1, num2str(nums)];
bg = repmat(perfMesh(:,1),1, numel(headerText));
h = figure; imagesc(bg, [0 1]);
colormap(gray);
cells = [...
    nTreeMesh(:,);
    minLeafSizeMesh(:,);
    NVarToSampleMesh(:,);
    class1PerfMesh(:,);
    class2PerfMesh(:,);
    avgPerfMesh(:,);
    perfMesh(:,);
    featCountMesh(:,);...
]
```matlab
131 cellsFirst = cells(:, 1:3);
textStrings1 = [num2str(cellsFirst(:)) repmat(' ', numel(cellsFirst), 1)];
133 cellsRemaining = cells(:, 4:end);
textStrings2 = [num2str(cellsRemaining(:) * 100, '%0.2f') repmat('%', numel(cellsRemaining), 1)];
textStrings3 = [textStrings1; textStrings2];
textStrings = cellstr(textStrings3);
137 [x, y] = meshgrid(1:size(cells, 2), 1:size(cells, 1));
139 hStrings = text(x(:), y(:), textStrings(:), ... 'HorizontalAlignment', 'center', 'FontSize', 18);
141 textColors = repmat(bg(:) < .5, 1, 3); % # Choose white or black for the
143 set(hStrings, {'Color'}, num2cell(textColors, 2)); % # Change the text colors
145 set(gca, 'xaxisLocation', 'top');
147 a = 1 - perfMesh(:,);
b = featCountMesh(:,);
149 b = NVarToSampleMesh(:,);
151 [c, idx] = sort(b);
d = a(idx);
155 figure; plot(c, d, '-' );
157 xlabel('Complexity ');
ylabel('Validation Error');
159 if (exist([keyword '.mat'], 'file'))
161 load(keyword);
else
165 rdfStruct = rdfTrain(trainFeatures, trainClasses,...
168 'nTree', nTreeMesh(idxIt), 'minLeafSize', minLeafSizeMesh(idxIt),... 
170 'NVarToSample', NVarToSampleMesh(idxIt));
172 save(keyword, 'rdfStruct');
end
174 testClasses(4) = [];
testFeatures(4) = [];
testClassPerfs = zeros(numel(testClasses), numel(classes)+1);
178 for idxTest = 1:numel(testClasses)
180 testClassesPicked = testClasses(idxTest);
182 testFeaturesPicked = testFeatures(idxTest);
184 testClassesPredict = rdfPredict(rdfStruct,...
186 for idxClass = 1:numel(classes)
188 classIdx = testClassesPicked==classes(idxClass);
190 testClassPerf = numel(find(testClassesPredict(classIdx)==... 
192 classes(idxClass))) / numel(find(classIdx));
194 testClassPerfs(idxTest, idxClass) = testClassPerf;
198 perf = mean(testClassesPredict==testClassesPicked);
testClassPerfs(idxTest, end) = perf;
end
197 function varargout = rdfPredict(varargin)
% RDFPREDICT
if (nargin<2)
 error(['Usage: perf = rdfPredict(model, features, classes, Option, Value,...
\newline Options: nTree, maxDepth, minVar, nFeat, nInst, kFold']);
else
 rdfStruct = varargin{1};
 featMat = varargin{2};
 classes = varargin{3};
end
```
Tree = rdfStruct.nTree;
postiers = 0;
for idxTree = 1:nTree
% Pick nFeat random features or nInst random instances
idxFeatsPicked = 0;
evalModelDescription = ['idxFeatsPicked = rdfStruct.idx' ...
20   num2str(idxTree) ';']
22   eval(evalModelDescription);
featPicked = featMat(:,idxFeatsPicked);
24   evalTxt = ['tree = rdfStruct.m' num2str(idxTree) '];'
26   eval(evalTxt);
28   posterior = predict(tree,featPicked);
postiers = postiers+posterior/nTree;
end
classesPred = round(postiers);
perf = mean(classesPred==classes);
varargout{1} = classesPred;
varargout{2} = perf;
end

function rdfStruct = rdfTrain(varargin)

% RDFTRAIN
if (nargin<2)
    error('Usage: rdfStruct = rdfTrain(%s, %s, %s, %s, %s, %s, %s, %s, ...
7   'features', 'classes', 'Option', 'Value', ...
8   ', 'n Options: nTree, minLeafSize, NVarToSample', ...')
end
featMat = varargin{1};
classes = varargin{2};

% Constants
nFeatures = size(featMat,2);
nInstances = size(featMat,1);

% Set defaults
nFeatToUse = floor(nFeatures);
nInstToUse = floor(nInstances);
nTree = 500;
minXValPerf = 0.7;
NVarToSample = 0;
minLeafSize = 1;

% maxDepth = inf;
kFold = 1;
for idxArg = 3:2:(nargin)
    text=varargin{idxArg};
    value=varargin{idxArg+1};
    if (strcmp(text, 'nTree'))
        nTree = value;
    elseif (strcmp(text, 'nFeat'))
        nFeatToUse = value;
    elseif (strcmp(text, 'nInst'))
        nInstToUse = value;
    elseif (strcmp(text, 'minLeafSize'))
        minLeafSize = value;
    elseif (strcmp(text, 'NVarToSample'))
        NVarToSample = value;
    elseif (strcmp(text, 'minXValPerf'))
        minXValPerf = value;
    elseif (strcmp(text, 'kFold'))
        kFold = value;
    else
        fprintf('Unknown option: %s',text);
        continue;
    end
    fprintf('%s set to %d',text,double(value));
end

% Cross validation after forest is formed to get weights? Is it too much?
rfStruct = trainForest(featMat, classes, nFeatToUse, nInstToUse, nTree, kFold, nFeatures, nInstances, minXValPerf,...
56   NVarToSample, minLeafSize);
end
function rdfStruct = trainForest ( featMat , classes , nFeatToUse , ... 
   nInstToUse , nTree , kFold , nFeatures , nInstances , minXValPerf , ... 
   NVarToSample , minLeafSize )

rdfStruct = struct ;
rdfStruct . nTree = nTree ;
rdfStruct . nFeatToUse = nFeatToUse ;
rdfStruct . nFeatures = nFeatures ;
rdfStruct . nInstToUse = nInstToUse ;
rdfStruct . nInstances = nInstances ;
rdfStruct . kFold = kFold ;
rdfStruct . minLeafSize = minLeafSize ;
rdfStruct . minXValPerf = minXValPerf ;
rdfStruct . NVarToSample = NVarToSample ;

idxTree = 1 ;
featuresSelected = [ ] ;

while ( idxTree <= nTree )
    % Pick nFeat random features or nInst random instances
    % generate decision tree
    featList = unique ( randi ( nFeatures , 1 , 20 * nFeatToUse ) , 'stable' ) ;
    idxFeatsPicked = featList ( 1 : nFeatToUse ) ;
    featPicked = featMat ( : , idxFeatsPicked ) ;
    % Save features used
    evalModelIdxFeats = [ 'rdfStruct . idx ' num2str ( idxTree ) '=idxFeatsPicked ; ' ] ;
    eval ( evalModelIdxFeats ) ;
    % Pick instances
    if ( nInstToUse ~= nInstances )
        idxInstList = crossvalind ( 'kfold' , classes , 20 ) ;
        idxInstPicked = idxInstList == 1 ;
        featPicked = featPicked ( idxInstPicked , : ) ;
        classesPicked = classes ( idxInstPicked ) ;
    else
        classesPicked = classes ;
    end
    % Cross-validation
    if ( kFold > 1 )
        idxXValList = crossvalind ( 'kfold' , classesPicked , kFold ) ;
    else
        idxXValList = zeros ( size ( classesPicked ) ) ;
    end
    bestModel = struct ;
    bestPerf = -1 ;
    for idxXVal = 1 : kFold
        idxTrain = idxXValList ~= idxXVal ;
        idxValid = idxXValList == idxXVal ;
        trainClassesPicked = classesPicked ( idxTrain ) ;
        trainFeaturesPicked = featPicked ( idxTrain , : ) ;
        validClassesPicked = classesPicked ( idxValid ) ;
        validFeaturesPicked = featPicked ( idxValid , : ) ;
        % Train tree
        if ( NVarToSample ~= 0 )
            tree = ClassificationTree . fit ( trainFeaturesPicked , trainClassesPicked , ... 
                'prior' , 'uniform' , 'NVarToSample' , NVarToSample , 'minleaf' , minLeafSize ) ;
        else
            tree = ClassificationTree . fit ( trainFeaturesPicked , trainClassesPicked , ... 
                'prior' , 'uniform' , 'minleaf' , minLeafSize ) ;
        end
        % Validate tree
        if ( kFold > 1 )
            classesPred = predict ( tree , validFeaturesPicked ) ;
            perf = mean ( classesPred == validClassesPicked ) ;
            if ( perf > bestPerf )
                bestPerf = perf ;
                bestModel = tree ;
            end
        else
            bestPerf = 1 ;
            bestModel = tree ;
        end
    end
    if ( bestPerf < minXValPerf )
        continue
    end
    strFeaturesUsed = char ( unique ( bestModel . CutVar ) ) ;
    idxIdxFeaturesUsed = str2num ( strFeaturesUsed ( : , 2 : end ) ) ; %#ok<ST2NM>
featuresUsed = idxFeatsPicked(idxIdxFeaturesUsed);
featuresSelected = [featuresSelected, featuresUsed]; %#ok<AGROW>
featuresSelected = unique(featuresSelected);
evalModel = ['rdfStruct.m' num2str(idxTree) '=bestModel; '];
eval(evalModel);
evalL = ['rdfStruct.L' num2str(idxTree) '=bestPerf; '];
eval(evalL);
idxTree = idxTree+1;
end
rdfStruct.FeaturesSelected = unique(featuresSelected);
end

function [classes, features] = readData(fileList)
% Read data
for i=1:size(fileList,1)
    if iscell(fileList)
        filename = fileList{i};
    else
        filename = fileList(i,:);
    end
    fid = fopen(filename, 'rt');
tline = fgets(fid);fid = fopen(filename, 'rt');
numData = numel(regexp(tline, '^[0-9]+', 'match'))-2;
idxList = num2str((1:numData));
idxList(:,end+1:end+4) = repmat('%f',numData,1);
evalStr = idxList';
rs = evalStr(idxList,:);
dataset = textscan(fid, ['%f ' evalStr], 'Delimiter', ',', 'CollectOutput',1, 'BufSize', 99999);
fclose(fid);
dataset2 = dataset{:};
if (size(fileList,1)>1)
    classes{i} = dataset2(:,1);
    features{i} = dataset2(:,2:end);
else
    classes = dataset2(1,1);
    features = dataset2(:,2:end);
end
end

function [h] = reportPerformances(mat, labels, filename)
% reportPerformances report performances
h = figure;imagesc(mat); %# Create a colored plot of the matrix values
pos = get(h, 'Position');
[nRows, nCols] = size(mat);
set(h, 'Position', [pos(1) pos(2) 650 400]);
set(gcf, 'PaperPositionMode', 'auto');
colormap(flipud(gray)); %# Change the colormap to gray (so higher values are
%# black and lower values are white)
textStrings = num2str(mat(:)*100,'%0.2f'); %# Create strings from the matrix values
textStrings = strtrim(cellstr(textStrings)); %# Remove any space padding
textContent = strcat(cellstr(textStrings),','); %# Create x and y coordinates for the strings
hStrings = text(x(:),y(:),textContent(:),...
    'HorizontalAlignment','center','FontSize',18);
midValue = mean(get(gca,'CLim'));
textContentColors = repmat(mat(:)>midValue,1,3); %# Choose white or black for the
%# text color of the strings so
%# they can be easily seen over
%# the background color
set(hStrings,{'Color'},num2cell(textColorColors,2)); %# Change the text colors
set(gca,'XTick',1:nCols,... %# Change the axes tick marks
    'XTickLabel', labels.X,...
    'YTick',1:nRows,...
    'YTickLabel', labels.Y,...
    'TickLength',[0 0]);
set(gca,'xaxisLocation','top')
saveas(gcf, filename, 'psc2');
References


