Randomized Greedy Algorithm

**Algorithm 1: Cost Sensitive RF**

1. **Input**: \( X \in \mathbb{R}^{m,n}, y \in \mathbb{Z}, C \in \mathbb{R}, \lambda \in \mathbb{R} \)
2. \( T \leftarrow T \)
3. for each tree \( i=1:N \) do
   1. Randomly sample training data to form \( X' \) and \( y' \)
   2. \( T, C' \leftarrow \text{GreedyTree}(X', y', C, \lambda) \)
   3. \( T \leftarrow T \cup T' \)
4. return \( T \)

**Algorithm 2: GreedyTree**

1. for each attribute \( i=1:M \) do
   1. Randomly sample splits \( s_j \) and compute \( F(s_j) = H(T) - H(T \setminus s_j) - \lambda C_j \), where \( H(T) \) is the information entropy
   2. \( s_j \leftarrow \arg\min_j F(s_j) \)
   3. Create new node using feature \( i \) and split value \( j \)

**Complementary Tree Training**

**Algorithm 3: Complementary Cost Sensitive RF**

1. \( T \leftarrow \emptyset \)
2. for each training example \( j=1:M \) do
   1. \( W_j = \frac{1}{y_j} \)
   2. \( C_j = 0 \)
   3. \( C \leftarrow \{C_i \} \)
   4. for each tree \( i=1:N \) do
      1. Randomly sample training data to form \( X' \) and \( y' \)
      2. \( T' \leftarrow \text{GreedyTree}(X', y', C, \lambda) \)
      3. \( T \leftarrow T \cup T' \)
3. return \( T \)

**Problem Statement**

Similar to the formulation in [2], our goal is to learn a classifier \( F \) from a family of functions \( \mathcal{F} \) that minimizes the sum of the expected errors and the computational cost of the final feature set:

\[
\min_{F \in \mathcal{F}} \mathbb{E}_{y \sim p(y \mid x)}[L(y, f(x))] + \lambda \mathbb{E}_{j} \left[ C(f, x) \right]
\]

where \( L(y, \hat{y}) \) is a loss function and \( C(f, x) \) is the cost of evaluating the function of \( f \) on example \( x \).

Our formulation differs from [2] in that we do not have a constraint on the feature costs, but rather incorporate the cost minimization into the objective itself. Since in practice we are given a training set, not a distribution, we will instead solve the following problem:

\[
\min_{f} \mathbb{E}_{x \sim p(x)}[L(y, f(x))] + \lambda \sum_{j=1,i \in S} C_j
\]

**Randomized Greedy Algorithm**

**Algorithm 2: GreedyTree**

1. for each attribute \( i=1:M \) do
   1. Randomly sample splits \( s_j \) and compute \( F(s_j) = H(T) - H(T \setminus s_j) - \lambda C_j \), where \( H(T) \) is the information entropy
   2. \( s_j \leftarrow \arg\min_j F(s_j) \)
   3. Create new node using feature \( i \) and split value \( j \)

**Results**

![Cost vs. accuracy tradeoff for three datasets. FFS - full feature set considered at each split; RFS - randomized feature set of size \( \sqrt{n} \) considered at each split; FFS + CT - full feature set and complementary training (Algorithm 3).](image)

**Datasets**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Num. Instances</th>
<th>Num. Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diabetes Diagnosis</td>
<td>768</td>
<td>8</td>
</tr>
<tr>
<td>Spam Filtering</td>
<td>4601</td>
<td>57</td>
</tr>
<tr>
<td>Digit Recognition</td>
<td>2000</td>
<td>240</td>
</tr>
</tbody>
</table>

**Feature Costs**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. Times Pregnant</td>
<td>1.00</td>
</tr>
<tr>
<td>Glucose Tolerance</td>
<td>17.61</td>
</tr>
<tr>
<td>Diastolic Pb</td>
<td>1.0</td>
</tr>
<tr>
<td>Triceps</td>
<td>1.0</td>
</tr>
<tr>
<td>Insulin</td>
<td>22.78</td>
</tr>
<tr>
<td>Mass Index</td>
<td>1.0</td>
</tr>
<tr>
<td>Pedigree</td>
<td>1.0</td>
</tr>
<tr>
<td>Age</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Selected References**


**Conclusions**

Here we show that a simple modification to the random forest algorithm allows for user control over the computational cost of the trained classifier. In order to account for the increased intra-forest correlation when more variables are considered at each node split, we also test a boosting-like iterative sample reweighting strategy (based on [3]), which generally improves performance. Overall, these results indicate that for several real world problems it is possible to significantly reduce test time cost with minimal effects on accuracy.

**Future Work**

- Instead of selecting features greedily while constructing forest splits, select features before building forest.
- Consider variations on standard wrapper and filter methods.
- In the case where common feature computational subroutines exist, treat as a submodular optimization problem, for which efficient approximation algorithms exist.
- At each split, optimize the number of variables to include in random subset, rather than using a fixed number.

**Acknowledgements**

Thank you to Junjie Qin for valuable feedback.