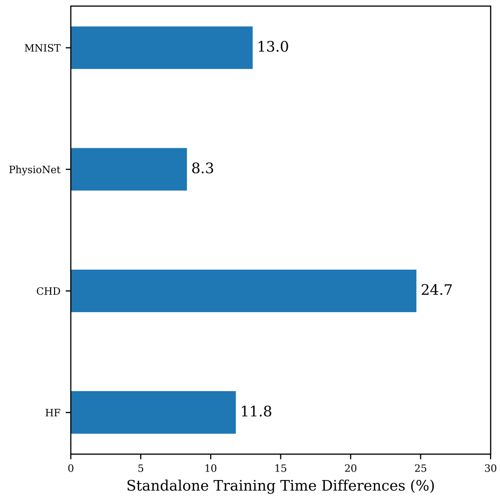

Figure A.1: Relative differences in the algorithm training times between the standalone and integrated implementations of the SO-MCTS.

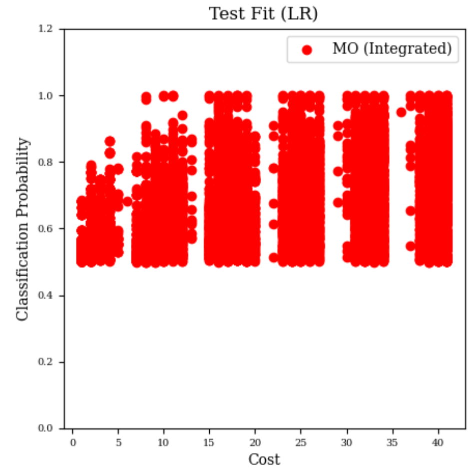

Figure A.2: Solutions for the MO-MCTS integrated implementation with the logistic regression classifier and fit strategy for the Heart Failure data set.

A Experiments

A.1 Experimental Results

Comparsion of the Standalone and Integrated Implementations

The F1 AUCs of the SO-MCTS integrated implementation shows relative improvement of 4.6% and 1.8% for the logistic regression and neural network classifiers from the SO-MCTS standalone implementation. Solutions in the objective space do not show differences between the two implementations. In Figure A.1, we show the relative difference in the algorithm training times for the standalone implementation from the integrated implementation, where the standalone implementation has faster relative algorithm training times by 8.3% and 24.7% from the integrated implementation. When the algorithm training time is another constraint in the usage of the MCTS algorithm for feature acquisition, it is advantageous to use the standalone implementation with lower training times if there is an option of slightly higher AUC.

Comparsion of the Classifier Strategies

For the Heart Failure, Coronary Heart Disease, and PhysioNet data sets, we use the fit strategy, where each subset of the feature set is used to train a single classifier. In comparison to the fit strategy, the best performing strategies with the SO-MCTS integrated implementation show relative performance improvements of 4.1% to 24.2%. For the MO-MCTS integrated implementation, the best performing strategies show relative improvements of 2.4% and 31.4%. We also plot the MO-MCTS solutions from the Heart Failure data set in the objective space in Figure A.2 for the logistic regression classifier with the fit strategy. In comparing the MO-MCTS solutions with the pretrain strategy, we observe that the solutions for the fit strategy are concentrated in the lower classification probability regions for all costs in Figure A.2. Thus, in the case when we use the MO-MCTS implementation for tight budget situations, it is also advantageous to use the fit strategy, as solutions can be obtained for lower costs with slight decreases in confidence thresholds.

Comparsion of the Strategies for the Unacquired Continuous Feature Values

As described in a previous section, we also optimize a function strategy for unacquired continuous feature values in the classifiers. The optimized hyperparameters are provided in the Appendix B.2. For the logistic regression classifiers, the quadratic cost function strategy has the highest train F1 AUCs for the Heart Failure, Coronary Heart Disease and PhysioNet data sets. For the neural network classifiers, the quadratic cost function strategy has the highest train F1 AUCs for the Heart Failure data set and constant function of 0 for the Coronary Heart Disease and PhysioNet data sets. Thus, it is advantageous to use the quadratic cost function strategies to set the values of unacquired continuous features.

B Experimental Setup

B.1 Network Architectures

The same network architectures are used for the neural network and convolutional neural network classifiers and policy and value networks in the algorithms, Table B.1 and B.2.

B.2 Algorithm Hyperparameters

The algorithm hyperparameters are presented in Tables B.3-B.6.

B.3 Continuous Unacquired Feature Values

We fitted four functions with quadratic maximum at 0 cost, quadratic minimum at full cost, linear, and constant. The choices are shown in Tables B.7-B.9.

Hyperparameter	Heart Failure	Coronary Heart Disease	PhysioNet
Feedforward1 Units	32	512	256
Activation	ReLU	ReLU	ReLU
Feedforward2 Units	16	256	128
Activation	ReLU	ReLU	ReLU
Feedforward3 Units	8	128	64
Activation	ReLU	ReLU	ReLU

Table B.1: Neural network architectures for classification and policy and value networks in the algorithms.

Layer	Hyperparameter	Value
	Filters	64
Conv1	Kernel	3
	Dilation	2
Activation	–	ReLU
Max Pooling	Pool	2
	Filters	128
Conv2	Kernel	3
	Dilation	2
Activation	–	ReLU
Max Pooling	Pool	2
	Filters	256
Conv3	Kernel	3
	Dilation	2
Activation	–	ReLU
Max Pooling	Pool	2
Final Layer	Units	512

Table B.2: MNIST convolutional neural network architecture for classification and feature acquisition policy.

Hyperparameter	Heart Failure	Coronary Heart Disease	PhysioNet	MNIST
Number of simulations	100	100	100	100
c	1.0	1.0	1.0	1.0
Update frequency	18	20	36	100
Optimizer	Adam	Adam	Adam	Adam
Learning rate	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Retrain frequency	54	180	324	10000

Table B.3: SO-MCTS hyperparameters.

Hyperparameter	Heart Failure	Coronary Heart Disease	PhysioNet	MNIST
Number of simulations	100	100	100	100
c	2.0	1.0	1.0	1.0
Update frequency	18	20	36	100
Optimizer	Adam	Adam	Adam	Adam
Learning rate	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Retrain frequency	18	20	36	100

Table B.4: MO-MCTS hyperparameters.

Hyperparameter	Heart Failure	Coronary Heart Disease	PhysioNet	MNIST
Episodes	100	100	100	100
Batch size	6	20	18	25
Update frequency	6	20	18	25
γ	0.5	0.99	0.999	0.99
ϵ-decay	0.99	0.99	0.99	0.5
Learning rate	10^{-6}	10^{-6}	10^{-6}	10^{-7}
Optimizer	Adam	Adam	Adam	Adam
Retrain frequency	108	360	684	12000

Table B.5: DQN hyperparameters.

Hyperparameter	Heart Failure	Coronary Heart Disease	PhysioNet	MNIST
Episodes	100	100	100	100
Clip parameter	0.2	0.2	0.2	0.2
GAE parameter	0.95	0.95	0.95	0.95
Entropy coefficient	0.01	0.01	0.02	0.02
Value function coefficient	1.0	1.0	1.0	1.0
Learning rate	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Optimizer	Adam	Adam	Adam	Adam
Retrain frequency	120	400	360	10000

Table B.6: PPO hyperparameters.

Algorithms	Unacquired Features (LR)	Unacquired Features (NN)
MO-MCTS Integrated	Quad Min at 41 with -70	Quad Min at 41 with -70
SO-MCTS Integrated	Quad Min at 41 with -70	Quad Min at 41 with -70
SO-MCTS Integrated	Quad Max at 0 with -50	Quad Min at 41 with -50
DQN	Quad Min at 41 with -50	Quad Min at 41 with -70
PPO-PG	Quad Max at 0 with -70	Quad Min at 41 with -70
PPO-AC	Quad Max at 0 with -90	Quad Min at 41 with -70

Table B.7: Heart Failure data set.

Algorithms	Unacquired Features (LR)	Unacquired Features (NN)
MO-MCTS Integrated	Quad Max at 0 with -50	0
SO-MCTS Integrated	Quad Min at 51 with -70	0
SO-MCTS Integrated	Quad Min at 51 with -70	0
DQN	Quad Max at 0 with -10	0
PPO-PG	Quad Min at 51 with -20	0
PPO-AC	Quad Max at 0 with -90	0

Table B.8: Coronary Heart Disease data set.

Algorithms	Unacquired Features (LR)	Unacquired Features (NN)
MO-MCTS Integrated	Quad Max at 0 with -50	0
SO-MCTS Integrated	Quad Min at 229 with -70	0
SO-MCTS Integrated	Quad Min at 229 with -70	0
DQN	Quad Min at 229 with -60	0
PPO-PG	Quad Min at 229 with -60	0
PPO-AC	Quad Min at 229 with -60	0

Table B.9: PhysioNet data set.

C Pseudocodes

Algorithm 1: Single-objective Monte Carlo Tree Search Functions

<u>function</u> **MCTS**(v,I)
  for iteration = 1,2,. . .,I **do**
    train(v)
  end for

<u>function</u> **train**(v)
  v_l = **select**(v)
  expand(v_l)
  reward = **simulate**(v_l)
  backprop(v_l,reward)

<u>function</u> **makeChild**(v, a)
  Obtain the feature by a in s of v to set s'
  Create node v' with s' where $a(v') = a$
  return v'

<u>function</u> **select**(v)
  while True **do**
    if v unexplored or terminal **do**
      return v
    end if
    $v \leftarrow \underset{v' \in C(v)}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{\ln N(v)}{N(v')}}$
  end while

<u>function</u> **expand**(v)
  for all unacquired actions $a \in A(v)$ **do**
    $v' \leftarrow$ **makeChild**(v, a)
    Add v' to $C(v)$
    Set $a(v') = a$
  end for

<u>function</u> **simulate**(v)
  reward = 0
  while v not terminal **do**
    Choose $a \in A(v)$ uniformly at random
    $v \leftarrow$ **make child**(v, a)
    reward $+= Q(v)$
  end while
  return reward

<u>function</u> **backprop**(v, reward)
  while v not null **do**
    $N(v) += 1$
    $Q(v) +=$ reward
    $v \leftarrow$ parent of v
  end while

Algorithm 2: Multi-objective Monte Carlo Tree Search (Integrated)

```
Input: Iteration number I, initial policy network weights θ, policy network update frequency f
Initialize policy network φ with θ
Initialize list L of visited nodes and their R and visit counts N
Initialize list M of global Pareto Front approximations P
i ← 0

function preprocess(L,M)
    Make each node v in L to be distinct with non-dominated union for R(v) and N(v) for duplicates
    A = 0⃗
    S = v in L
    for v in L do
        for action in A do
            Find child nodes of v in L
            for node in child nodes do
                R(node) = [R(node),M]
                A(action) ← A(action) + HV(R(node))
            end for
        end for
    Normalize A with division by max(A)
    return S, A

for sample = 1,2,...,m do
    i ← i + 1
    Initialize state s_0
    Initialize global Pareto Front approximation P
    Create root node v_0 with s_0
        R(v_0): local Pareto Front approximation
        N(v_0): visit count of v_0
        C(v_0): children of v_0
        a(v_0): action of v_0
    while v_0 not terminal do
        MO-MCTS(v_0,I)
        a ← φ_θ(s_0)
        v_0 ← makeChild(v_0, a)
    end while
    Append R(v) and N(v) to L
    M ← findGlobalP(M,P)
    if f % i == 0 do
        S, A ← preprocess(L,M)
        Train φ_θ on S and A
    end if
end for

function MO-MCTS(v,I)
    for iteration = 1,2,...,I do
        train(v)
    end for

function train(v)
    v_l = select(v)
    expand(v_l)
    reward = simulate(v_l)
    backprop(v_l,reward)

function makeChild(v, a)
    Obtain the feature by a in s of v to set s'
    Create node v' with s' where a(v') = a
    return v'

function HV(R(v))
    Set reference point at [−1.0, 0.0]
    hv = 0
    for front in R(v) do
        h = front[i][0] - reference[0]
        hv ← hv+ (front[i][1] - front[i − 1][1])h
    return hv

function select(v)
    while True do
        if v unexplored or terminal do
            return v
        end if
        for v' ∈ C(v) do
            R(v') ← R(v')/N(v') + c·sqrt(2 ln N(v) / N(v'))
        v ← argmax_{v' ∈ C(v)} HV(R(v'))
    end while

function backprop(v, reward)
    while v not null do
        N(v) ← N(v) + 1
        R(v)[0] ← R(v)[0]+reward[0]
        R(v)[1] ← R(v)[1]+reward[1]
        P ← findGlobalP(P, R(v'))
        v ← parent of v
    end while
```