Appendix

Data and statistical code for reproduction of the methods discussed in the main text and this Appendix are available at: https://github.com/sanjaybasu/MLforPMHD

Implementation of the machine learning algorithms was conducted using the h2o package in R.

Below, we provide further mathematical background on the implementation of each machine learning method discussed in the main text and provide details to enable interpretation and reproduction of the statistical code available at the above link.

<2>*Regularization*

Elastic net regularization seeks to minimize a penalty function over a grid of values for the regularization parameter λ, which controls the strength of the balancing parameter α between lasso regression (L1 regularization, which picks one correlated covariate and removes the others from the equation) and ridge regression (L2 regularization, which shrinks coefficients of correlated covariates towards each other). Specifically, for a logistic regression, the approach seeks to minimize the negative binomial log-likelihood expressed in (Eq. 1) [29]:

(Eq. 1)

$$\min_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} -\left[\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \left(\beta_0 + x_i^T\beta\right) - log(1 + e^{(\beta_0 + x_i^T\beta)}\right] + \lambda\left[\frac{(1-\alpha)\left|\left|\beta\right|\right|_2^2}{2} + \alpha\left|\left|\beta\right|\right|_1\right]$$

The regularization algorithm performs coordinate descent on a quadratic approximation to the log-likelihood. We trained the GLM estimators to find the optimal λ value per the above equation, at each α value, across internal cross-validations on the training dataset.

<2> *Gradient Boosting Trees*

Classification trees were developed with a gradient boosting machine (GBM), an ensemble learning algorithm that sequentially builds classification trees on all features of each derivation dataset.[6,7] Classification trees are developed by sequentially selecting covariates to parse each derivation dataset into subsets with high between-group and low within-group variance in the probability of the primary outcome. The gradient boosting seeks to improve the choice of covariates and their combinations over time to minimize error between observed and predicted outcome rates in each terminal node, while repeatedly sampling from the underlying data to prevent overfitting. Specifically, the algorithm we use follows the common procedure below (Eqs. 2 through 7) to build k regression trees with incremental boosts of the function f over M iterations[3]:

(Eq. 2) Initialize $f_{k0} = 0$, k = 1, 2, …, K

(Eq. 3) For m = 1 to M, compute the gradient $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}$ for all k = 1, 2, …, K

(Eq. 4) For k = 1 to K, fit a tree to targets $r_{ikm} = y_{ik} - p_k(x_i)$, for i = 1, 2, …, N, producing terminal regions R_{jkm}, 1, 2, …, J_m

(Eq. 5) Calculate the step $\gamma_{jkm} = \frac{K-1}{K} = \frac{\sum_{x_i\in R_{jkm}}(r_{ikm})}{\sum_{x_i\in R_{jkm}}|r_{ikm}|(1-|r_{ikm}|)}$, j = 1, 2, …, J_m

(Eq. 6) Update the function $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m}\gamma_{jkm}I(x\in R_{jkm})$

(Eq. 7) Output $\hat{f}_k(x) = f_{kM}(x)$

The GBM requires tuning and can be sensitive to hyperparameter choices. Hence, we varied the key hyperparameters across broad ranges while training multiple

GBMs. Specifically, we trained GBM estimators with common values of the learning rate, the maximum tree depth, and column sample rate. The learning rate or shrinkage controls the weighted average prediction of the trees and refers to the rate (ranging from 0 to 1) at which the GBMs change parameters in response to error (ie, learn) when building the estimator. Lower learning rate values enable slower learning but require more trees to achieve the same level of fit as with a higher learning rate. Lower learning rates also help to avoid overfitting despite being more computationally expensive. The maximum tree depth refers to the number of layers of the three, thus controlling how many splits can occur to separate the population into subpopulations. In the algorithm we used, a tree will no longer split if either the maximum depth is reached or a minimum improvement in prediction (relative improvement in squared error reduction for a split $>1^{-5}$) is not satisfied. While deeper trees can provide more accuracy on a derivation dataset, they can also lead to overfitting [3]. The sample rate (ranging from 0 to 1) refer to the fraction of the data sampled among columns of the data for each tree (such that 0.7 refers to 70% of the data sampled), which helps improve sampling accuracy. Values less than 1 for the sampling rate can help improve generalization by preventing overfitting. We use stochastic gradient boosting [30], such that to fit each GBM estimator, in each learning iteration a subsample of training data was drawn at random without replacement.

<2> Random Forest

As with the GBMs, random forests were constructed from classification trees, with a broad range of hyperparameter starting values entered into a random grid search. The algorithm applies a modified version of bootstrap aggregating (bagging) in which the tree learning algorithm selects, at each candidate split in tree learning process, a random subset of the covariates to build and train the tree (feature bagging). Trees are trained in parallel and split until either the maximum depth is reached or a minimum improvement in prediction (relative improvement in squared error reduction for a split $>1^{-5}$) is not satisfied. We can fit forests with varied values of the maximum tree depth, and column sampling rates. Note that random forest generally utilizes greater maximum tree depth and samples from a smaller number of covariates in each tree than GBM. The greater depth enables random forest to be more efficient on longer datasets, while GBM is more efficient on wider datasets [3]. By randomizing tree building and combining the trees together, random forest helps to reduce overfitting. A simple majority vote of the resulting trees' predictions produce an individual's final prediction from the forest of underlying trees.

<2> Deep Learning

Our deep learning algorithm first consisted of a multi-layer, feedforward network of neurons, where each neuron takes a weighted combination of input signals as per (Eq. 8):

(Eq. 8)

$$\alpha = \sum_{i=1}^{n} w_i x_i + b$$

reflecting a weighted sum of the input covariates with additional bias b, which is the neuron's activation threshold.[15] The neuron then produced an output $f(\alpha)$ that represents a nonlinear activation function. In the overall neural network, an input layer matches the covariate space, and is followed by multiple layers of neurons to produce abstractions from the input data, ending with a classification layer to match a discrete set of outcomes. Each layer of neurons provides inputs to the next layer, and the weights and bias values determine the output from the network. Learning involves adapting the weights to minimize

the mean squared error between the predicted outcome and the observed outcome across all individuals in the derivation dataset.

We developed deep learning estimators by training across various activation functions and hidden layer sizes. The activation functions included common maxout, rectifier, and tanh. The maxout activation function is a generalization of the rectified linear function given by equation (Eq. 9)[31]:

(Eq. 9)

$$f(\alpha) = max(0, \alpha), f(\cdot) \in \mathbb{R}_+$$

In maxout, each neuron chooses the largest output of 2 input channels, where each input channel has a weight and bias value. Users can include a version with "dropout", in which the function is constrained so that each neuron in the network suppresses its activation with probability P (<0.2 for input neurons, and <0.5 for hidden neurons), which scales network weight values towards 0 and forces each training iteration to train a different estimator, enabling exponentially larger numbers of estimators to be averaged as an ensemble to prevent overfitting and improve generalization.[32,33] The rectifier is a case of maxout where the output of one channel is always 0.[31]

<2> *Estimator Stacking and Ensemble Prediction.*

After training estimators using all combinations of the hyperparameters for each of the estimator types on each derivation dataset, we selected the weights and ensemble predictions for each derivation dataset using a GBM meta-learner algorithm.[21] The meta-learner applies the GBM principle to define the weight given to each of the base learners. The final ensemble prediction for each simulated individual from the machine learning approach was given by the weighted average of the predictions from each base learner to maximize the C-statistic in repeated cross-validation samples from the training data.

Appendix Table 1. Confusion matrix from the ensemble of learners, applied to the test dataset.

	True outcome		
Predicted outcome	**Had outcome**	**Did not have outcome**	
Predicted to have outcome	2378 true positives	698 false positives	Positive predictive value = 2378/(2378+698) = 0.77
Predictive to not have outcome	710 false negatives	2355 true negatives	Negative predictive value = 2355/(2355+710) = 0.77
	Sensitivity = 2378/(710+2378) = 0.77	Specificity = 2355/(2355+698) = 0.77	