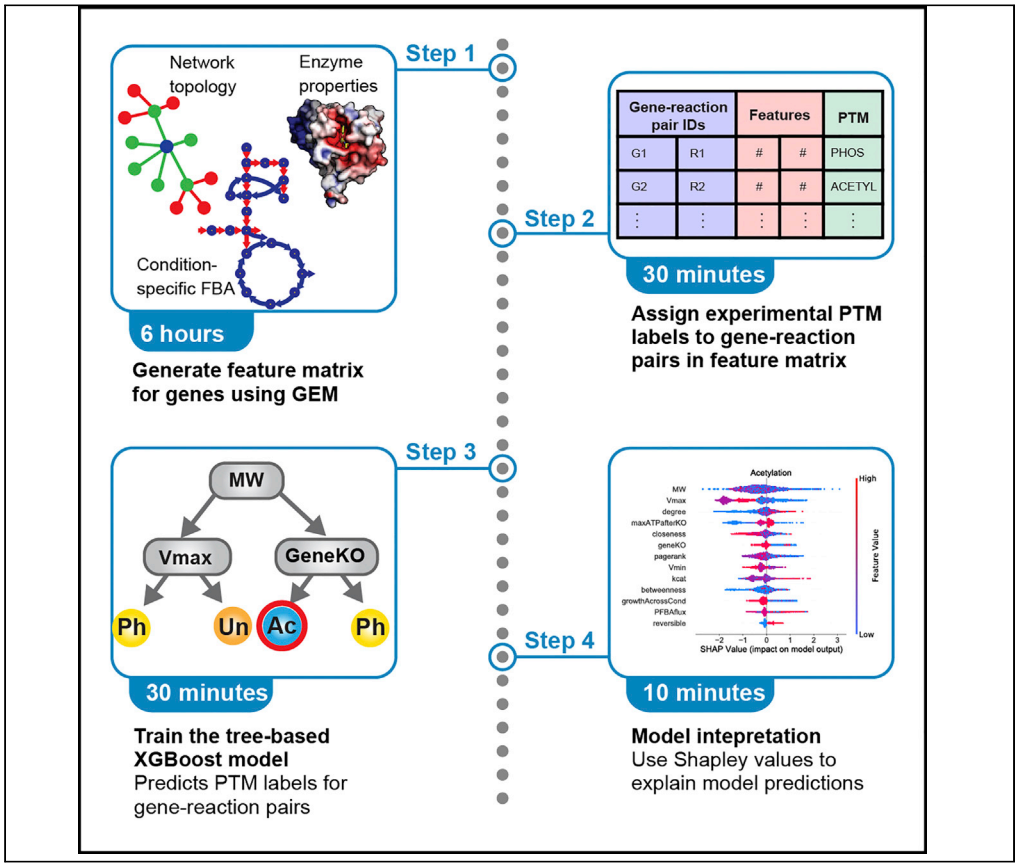
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for CAROM: A machine learning tool to predict post-translational regulation from metabolic signatures



This protocol describes CAROM, a computational tool that combines genome-scale metabolic networks (GEMs) and machine learning to identify enzyme targets of post-translational modifications (PTMs). Condition-specific enzyme and reaction properties are used to predict targets of phosphorylation and acetylation in multiple organisms. CAROM is influenced by the accuracy of GEMs and associated flux-balance analysis (FBA), which generate the inputs of the model. We demonstrate the protocol using multi-omics data from *E. coli*.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Kirk Smith, Nicole Rhoads, Sriram Chandrasekaran

kirksmi@umich.edu (K.S.)
csriram@umich.edu (S.C.)

Highlights

CAROM uses condition-specific enzyme and reaction properties to predict PTMs

Proteomics data and GEMs are used to train and tune XGBoost models

Construct PTM models for any organism with available GEM and omics data

Explain CAROM's predictions using Shapley values

Smith et al., STAR Protocols 3, 101799
December 16, 2022
https://doi.org/10.1016/j.xpro.2022.101799

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol for CAROM: A machine learning tool to predict post-translational regulation from metabolic signatures

Kirk Smith,[1,4,*] Nicole Rhoads,[1,2] and Sriram Chandrasekaran[1,3,5,*]

[1]Department of Biomedical Engineering, University of Michigan, Ann Arbor, MI 48109, USA

[2]Department of Pharmacology, University of Michigan, Ann Arbor, MI 48109, USA

[3]Center for Computational Medicine and Bioinformatics, University of Michigan, Ann Arbor, MI 48109, USA

[4]Technical contact: kirksmi@umich.edu

[5]Lead contact

*Correspondence: kirksmi@umich.edu (K.S.), csriram@umich.edu (S.C.)
https://doi.org/10.1016/j.xpro.2022.101799

SUMMARY

This protocol describes CAROM, a computational tool that combines genome-scale metabolic networks (GEMs) and machine learning to identify enzyme targets of post-translational modifications (PTMs). Condition-specific enzyme and reaction properties are used to predict targets of phosphorylation and acetylation in multiple organisms. CAROM is influenced by the accuracy of GEMs and associated flux-balance analysis (FBA), which generate the inputs of the model. We demonstrate the protocol using multi-omics data from *E. coli*.

For complete details on the use and execution of this protocol, please refer to Smith et al. (2022).

BEFORE YOU BEGIN

Post-translational modifications (PTMs) such as phosphorylation and acetylation are highly conserved mechanisms for regulation of cellular metabolism. However, the partitioning of regulation among the different mechanisms is unclear. The huge diversity of PTMs make experimental methods alone insufficient to understand their individual roles in metabolic regulation. To differentiate the unique roles of each PTM, a data-driven approach called Comparative Analysis of Regulators of Metabolism (CAROM) was developed (Smith et al., 2022). CAROM compares the properties of PTM targets including essentiality, flux, molecular weight, and topology.

A list of protein targets of each regulatory mechanism is input to CAROM, and their properties are analyzed using a genome-scale metabolic model (GEM). CAROM compares 13 properties (Table 1) of each metabolic reaction to predict condition-specific PTM regulation sites. These properties were chosen based on the hypothesis that target preferences of PTMs can be inferred from biochemical, topological and flux properties of the targets. This hypothesis stems from prior experimental and computational flux studies that have found an association between regulation by phosphorylation and changes in reaction fluxes (Oliveira et al., 2012). By applying CAROM to multi-omics data in yeast (Treu et al., 2014; Weinert et al., 2014; Murphy et al., 2015) and *E. coli* (Soares et al., 2013; Weinert et al., 2013; Houser et al., 2015), we showed that essentiality and flux are useful features to predict regulation of different enzymes across different species.

CAROM was linked to a machine learning algorithm to create CAROM-ML, thereby allowing for further analysis of the 13 key features' influence on regulation. CAROM-ML was applied to well characterized cellular processes in yeast, *E. coli*, and HeLa cells (Olsen et al., 2010; Kori et al., 2017). Each gene-reaction pair is assigned a class corresponding to PTM type (phosphorylated, acetylated, or

Table 1. Definitions of the thirteen features used by CAROM to compare and classify the targets of metabolic regulation

Feature number	Feature name	Definition
1.	pFBA flux	Fluxes from parsimonious flux balance analysis (pFBA). pFBA eliminates futile cycles and redundancy by minimizing total flux through the network while maximizing biomass.
2.	Gene KO	The maximum biomass/growth rate after gene knockout.
3.	Max ATP after KO	The maximum ATP generated after gene knockout.
4, 5.	Vmax, Vmin	Flux variability analysis (FVA) is used to find the maximum and minimum flux through each reaction that still satisfies the maximum biomass growth objective.
6.	Growth across conditions	A measure of organism viability across 87 different conditions. Gene knockout is performed for each condition and the number of times a gene is found to be lethal is used as the metric.
7.	Closeness	Inverse sum of the distance from a node to all other nodes.
8.	Betweenness	Measures how often each graph node appears on a shortest path between two nodes in the graph.
9.	Page rank	Results from a random walk of the network. The score is the average time spent at each node during the random walk.
10.	Degree	Number of edges connecting to each node. A self-loop counts as two edges connecting to the node.
11.	Reversible	A binary variable (0 or 1) indicating whether the reaction is reversible.
12.	MW	Molecular weight of the enzyme
13.	Kcat	Catalytic activity of the enzyme

unknown). CAROM-ML classifies PTM target proteins and proved to be highly accurate (AUC > 0.8, MCC > 0.8, R > 0.6) across different conditions and species.

CAROM-ML was built with the Python XGBoost package. The XGBoost algorithm uses boosted decision trees to minimize the error of the previous learner and prevent overfitting of the model (Chen and Guestrin, 2016). SHapley Additive exPlanation (SHAP) was used to interpret the results from CAROM-ML and to quantify the contribution of each feature to the accuracy of the model (Lundberg et al., 2020). SHAP calculates the contribution from each feature to the ML model using the game theory concept of Shapley values. Interpretation of SHAP values for all observations can clarify overall feature importance, how a feature impacts model output, and relationships between the predictor features.

Overall, this approach can identify regulatory mechanisms dominant in different parts of the metabolic network to improve accuracy of GEMs, and ultimately develop novel drug targets or direct metabolic engineering efforts. This approach can lead to the development of more complete next-generation GEMs with multiple regulatory mechanisms (Chung et al., 2021).

The subsequent section details the computational requirements and installation steps that should be completed prior to using CAROM-ML. Users should be able to use this same set of instructions to setup and run CAROM-ML whether using Windows or MacOS.

MATLAB

Timing: 1 h

1. The input data for CAROM-ML is generated and processed in MATLAB, primarily because we require the COBRA toolbox (see next step). MATLAB can be downloaded at this link.

COBRA toolbox

Timing: 30 min

2. COBRA (COnstraint-Based Reconstruction and Analysis) is a MATLAB toolbox used for constraint-based modeling of biochemical networks (Heirendt et al., 2019). Detailed instructions on setting up COBRA for MATLAB are found here: https://opencobra.github.io/cobratoolbox/stable/installation.html.

Note: This toolbox is also available in Python (https://opencobra.github.io/cobrapy/).

Python and anaconda

Timing: 1 h

3. Python will be required to run the machine-learning code. For your Python platform, we recommend installing Anaconda, a package and environment manager which also provides applications for running code. Anaconda can be downloaded at https://www.anaconda.com/products/individual.

Note: Python will automatically be installed along with Anaconda. We recommend installing the Python 3.8 version to avoid potential conflicts with the CAROM-ML, which has not yet been tested with newer versions.

CAROM files

Timing: 5 min

4. Download the files required for running CAROM-ML, which can be found under the *Files* tab on the project's Synapse page: https://www.synapse.org/#!Synapse:syn20843407/wiki/597068 (alternatively use https://doi.org/10.7303/syn20843407).
   a. After downloading the zipped file, unzip it within your project directory, which we will call *project_folder*.

Note: The multi-organism dataset used by Smith et al. is included. As later detailed in the step-by-step method details section, the CAROM-ML input matrix contains thirteen features, along with a column for the PTM labels. These features are defined in Table 1.

Setting up your Python virtual environment

Timing: 10 min

5. Create a virtual environment using the *environment.yml* file, which should have been downloaded from Synapse into your project folder. This file contains all of the Python dependencies required to run CAROM.
   a. Navigate to the project folder using the command prompt or terminal:
   (Windows/MacOS)

```
>cd Desktop/project_folder
```

   b. Run the following command to create a virtual environment (here we call the environment "conda-env", however you may name it anything):

(Windows/MacOS)

```
>conda env create -prefix "./conda-env" -file environment.yml
```

You should now be ready to run CAROM-ML.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
CAROM supplementary data	Smith et al. (2022)	iScience main text: Supplementary material
E.coli acetylomics	Weinert et al. (2013)	Ecoli_Acetyl_Weinert.xlsx - syn34542449 - Files (synapse.org)
E.coli phosphoproteomics	Soares et al. (2013)	Ecoli_Phos_Soares.xlsx - syn34542450 - Files (synapse.org)
E.coli genome-scale metabolic model	Orth et al. (2011)	Ecoli_GEM_Orth_iJO1366.mat - syn34539904 - Files (synapse.org)
E.coli KEGG to NCBI map	N/A	Ecoli_bnum_to_enzyme_map.xlsx - syn34542574 - Files (synapse.org)
Software and algorithms		
MATLAB	N/A	https://www.mathworks.com/products/matlab.html
Anaconda	N/A	https://www.anaconda.com/products/individual
COBRA	Heirendt et al., (2019). "Creation and Analysis of Biochemical Constraint-Based Models Using the COBRA Toolbox v.3.0." *Nature Protocols*	https://opencobra.github.io/cobratoolbox/stable/
CAROM-ML source code	Smith et al. (2022)	carom.py - syn25792906 - Files (synapse.org)
CAROM-ML tutorial code	Smith et al. (2022)	carom_test_install.py - syn25886762 - Files (synapse.org)
Conda environment file	Smith et al. (2022)	environment.yml - syn25876213 - Files (synapse.org)

STEP-BY-STEP METHOD DETAILS

Here we describe the step-by-step methods for running CAROM-ML, from generating the input data in MATLAB to running the source code in Python. This tutorial will use *E.coli* acetylation data from the Weinert et al. study and phosphorylation data from Soares et al. The major steps of the data workflow are summarized in Figure 1.

Generating the feature matrix for CAROM-ML

Timing: ~6 h

CAROM-ML uses thirteen features to predict the PTM applied to each gene-reaction pair. As detailed in Table 1, these predictors are related to metabolic flux, gene essentiality, network connectivity and enzymes properties. Here we will demonstrate how to generate this matrix of *N* gene-reaction pairs x 13 features.

1. Download the Orth et al. *E.coli* GEM from the CAROM-ML Synapse page or the BiGG database (http://bigg.ucsd.edu/models/iJO1366) and load into MATLAB.

   Note: It is important that the user selects the MATLAB file (with .mat extension) from the BiGG download page.

2. Load the GEM into MATLAB, use the *load* function.
(MATLAB)

```
>ecoli_GEM=readCbModel('Ecoli_GEM_Orth_iJO1366.mat')
```

   Note: It is important to constrain the GEM as necessary in order to better simulate the condition of interest. For modeling different metabolic states, experimental time-course

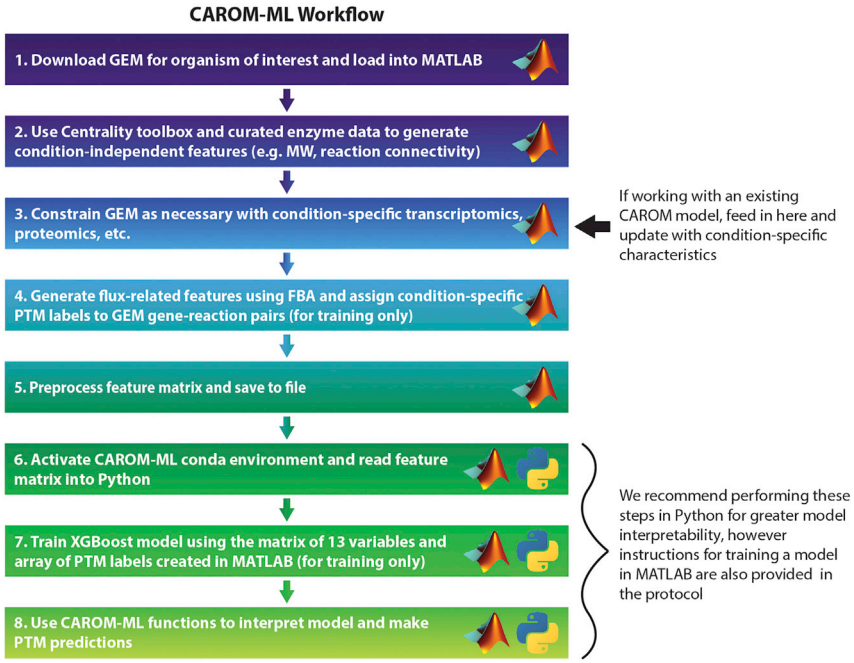


Figure 1. Workflow diagram summarizing the steps involved in creating a CAROM-ML model

metabolomics can be used with the Dynamic Flux Activity (DFA) algorithm (Chandrasekaran et al., 2017; Campit and Chandrasekaran, 2020). For simulating other experimental conditions from transcriptomics or proteomics, a list of up- and down-regulated genes/proteins can be used as inputs to determine fluxes (Shen et al., 2019). Due to the various techniques that can be used for adjusting the GEM, this is not covered in this protocol.

3. Generate the topological variables of the feature matrix.
   a. Use the Centrality Toolbox to calculate *closeness*, *betweenness*, *page rank* and *degree*. The *topo_rxns* matrix should therefore be *N* reactions x 4 topological features.

(MATLAB)

```
>[groups, orphans, R, C] = connectedComponents(ecoli_GEM);

>GG = graph(R);

>topo_rxns(:,1) = centrality(GG,'closeness');

>topo_rxns(:,2) = centrality(GG,'degree');

>topo_rxns(:,3) = centrality(GG,'betweenness');

>topo_rxns(:,4) = centrality(GG,'pagerank');
```

4. Extract reaction reversibility information from the GEM.

(MATLAB)

```
>reversible = ecoli_GEM.rev;
```

5. Use FBA (Orth et al., 2010) to calculate the *geneKO* and *maxATPafterKO* features by knocking out each gene in the model one at a time and solving for the optimization problem for each iteration.

(MATLAB)

```
% initialize the COBRA toolbox and change solver to 'gurobi'
initCobraToolbox(false)
changeCobraSolver('gurobi', 'all');
% initialize variables
geneKO = NaN(length(ugenes),1);
maxATPafterKO = NaN(length(ugenes),1);
% find location of ATP in GEM metabolite list
ix_ATP = ismember(ecoli_GEM.metNames, 'ATP');
for i = 1:length(ugenes)
  % knockout gene 'i' and solve model
  GEM_temp = deleteModelGenes(ecoli_GEM, ugenes(i));
  solution = optimizeCbModel(GEM_temp,[],'one'); % solve FBA problem
  geneKO(i) = solution.f;
  % change objective to ATP and resolve
  GEM_atp = addDemandReaction(GEM_temp, ecoli_GEM.mets(ix_ATP));
  GEM_atp = changeObjective(GEM_atp, GEM_atp.rxnNames(end));
  solution = optimizeCbModel(GEM_atp,[],'one');
  maxATPafterKO(i) = solution.f;
end
```

6. Use flux variance analysis to derive a range of fluxes, *Vmin* and *Vmax*, through every reaction in the network (Mahadevan and Schilling, 2003).

(MATLAB)

```
>[Vmin, Vmax] = fluxVariability(ecoli_GEM, 100, 'max', ecoli_GEM.rxns, 1, true);
```

7. Calculate the 'growth across conditions' values for each gene in the GEM.

Note: The 87 different conditions that are tested are controlled using exchange reactions.

(MATLAB)

```
% find exchange rxns in GEM
load carom_GEMs ecoli_exchanges
[ix, excpos] = ismember(ecoli_exchanges, ecoli_GEM.rxnNames);
```

```
exc_pos = exc_pos(exc_ix)
% remove glucose and glutamine
GEM_temp = ecoli_GEM
GEM_temp.lb(ismember(ecoli_GEM.rxnNames, {'D-Glucose exchange'}))=0;
GEM_temp.lb(ismember(ecoli_GEM.rxnNames, {'L-Glutamine exchange'}))=0;
% initialize growth rate matrices
condition_wt1 = zeros(length(exchanges), 1);
geneko_biom_allcond = zeros(length(ugenes), length(ecoli_exchanges));
```

a. Loop through the exchange reactions, each time setting the LB for the current reaction to -1, and then run gene knockout for all genes.

(MATLAB)

```
for i = 1:length(exc_pos)
  % set lower bound of current exchange rxn to -1, then solve
  modeltemp1 = GEM_temp;
  modeltemp1.lb(exc_pos(i)) = -1;
  ffgeneko = optimizeCbModel(modeltemp1);
  condition_wt1(i,1) = ffgeneko.f;
  % do gene KO for each j
  for j = [1:length(ugenes)]
    modeltemp2 = deleteModelGenes(modeltemp1,ugenes(j));
    ffgeneko1 = optimizeCbModel(modeltemp2);
    geneko_biom_allcond(j,i) = ffgeneko1.f; % rows=genes, cols=conditions
    disp([j i])
  end
end
% Anything that reduces by 1% is condition-specific essential.
growth_norm = geneko_biom_allcond./condition_wt1';
growth_norm1 = (growth_norm < 0.99);
growthAcrossCond = sum(~growth_norm1,2)/length(exchanges); % fraction of conditions where
essential
```

8. Perform parsimonious flux balance analysis (pFBA), which maximizes growth rate while minimizing the sum of fluxes.

(MATLAB)

```
>solution = optimizeCbModel(ecoli_GEM, [], 'one');
>PFBAflux = solution.x
```

9. Retrieve the enzyme-specific properties, *MW* and *kcat*, for each gene in the GEM. These features will need to be gathered from biochemical databases like SABIO or BRENDA.

   Note: In some cases, this data is provided with the GEMs and simply need to be extracted from the GEMs for specific genes of interest.

10. Identify the significantly regulated enzymes using one or more omics datasets of interest. In Smith et al. (2022), acetylomics and phosphoproteomics data from various growth conditions were used for training.

    Note: It is up to the user to decide how the significantly regulated enzymes will be identified, whether that involves using fold change, z-scores or some other method. . For the CAROM study, fold change was used to label the significantly regulated proteins and their corresponding genes.

(MATLAB)

```
% load acetyl data w/ log2fc

ec_acet_data = readtable(''Ecoli_Acetyl_Weinert.xlsx'');

% load bnums-to-enzyme map for ecoli acetyl data

ec_bnums_to_enzymes = readtable("Ecoli_bnum_to_enzyme_map.xlsx");

% extract high and low acetyl genes

bnum_acet_high = string(ec_bnums_to_enzymes{ec_acet_data{:,4} > 2, "bnum"});

bnum_acet_low = string(ec_bnums_to_enzymes{ec_acet_data{:,4} < 0.5, "bnum"});

% load phos data w/ log2fc and gene names

phos_data = readtable(''Ecoli_Phos_Soares.xlsx'', 'Sheet','p-sites');

expression = '[b].\d*'; % extract KEGG IDs (start w/ 'b', followed by #)

match = string(regexp(phos_data.Gene_Names, expression, 'match','once'));

bnum_phospho_high = string(match(phos_data.T1_Normalized_by_Protein > 2));

bnum_phospho_low = string(match(phos_data.T1_Normalized_by_Protein < 0.5));
```

11. Create the final feature matrix.
    a. Map the gene-specific and reaction-specific features to the appropriate rows of the feature matrix, which are gene-reaction pairs.

    Note: Here we do not show *MW* and *kcat*, however the same procedure would follow once the user has collected those values from outside sources.

    (MATLAB)

```
%%% map features and PTM labels to gene-reaction pairs %%%

% geneKO, maxATP and growth across conditions

[ix pos] = ismember(rxntable_bnums(:,1), ugenes); sum(ix)

features(:,1) = geneKO(pos,1);

features(:,2) = maxATPafterKO(pos,1);
```

```
features(:,3) = growthAcrossCond(pos,1);

% fva, pfba and reversibility

[ix pos] = ismember(rxntable_bnums(:,2), ecoli_GEM.rxns);

features(:,4) = PFBAflux(pos,1);

features(:,5) = Vmin(pos,1);

features(:,6) = Vmax(pos,1);

features(:,7) = reversible(pos,1);

% topology

[ix pos] = ismember(rxntable_bnums(:,2), ecoli_GEM.rxns); sum(ix) %

features(:,8:11) = topo_rxns(pos,:);

feat_names = {'geneKO', 'maxATPafterKO', 'growthAcrossCond', 'PFBAflux', 'Vmin', 'Vmax',
'reversible', 'closeness', 'degree', 'betweenness', 'pagerank'};

% create table and add gene+rxn names

tbl_ecoli = array2table(features,'VariableNames',feat_names);

tbl_ecoli = addvars(tbl_ecoli,rxntable_bnums(:,1),rxntable_bnums(:,2),...

  'Before','geneKO','NewVariableNames',{'genes','rxns'});
```

b. Map the significantly regulated enzyme to the gene-reaction pairs. The result should be an array of binary values for each type of PTM.

Note: Any genes from the GEM that either do not meet the regulation threshold or are not found within the omics dataset should be assigned to the "unknown regulation" class.

(MATLAB)

```
% PTM labels

ec_acetyl = unique([bnum_acet_high; bnum_acet_low]);

ec_phos = unique([bnum_phospho_high; bnum_phospho_low]);

% 0=unknown, 1=acetylated

ix = ismember(rxntable_bnums(:,1), ec_acetyl); sum(ix)

tbl_ecoli.Acetyl = ix;

% repeat for phosphorylation

ix = ismember(rxntable_bnums(:,1), ec_phos); sum(ix)

tbl_ecoli.Phos = ix;
```

c. If the user wishes to examine these two PTM types together as multi-class problem, another column can be added which includes all three classes (acetylation, phosphorylation and unknown).

Note: In this case, the user must decide how to handle overlapping labels. In Smith et al. (2022), overlapping labels were assigned to the least represented class, phosphorylation.

(MATLAB)

```
% create multi-class target variable (Acetyl=0, Unknown=1, Phos=2)

tbl_ecoli.PTM = ones(length(tbl_ecoli.Acetyl),1);

tbl_ecoli.PTM(tbl_ecoli.Acetyl==1)=0;

tbl_ecoli.PTM(tbl_ecoli.Phos==1)=2;
```

12. Additional preprocessing steps are up to the user. Below we detail the primary data transformation steps performed for the CAROM paper.
    a. Fill in any missing feature data with the median value from its respective dataset. This should only apply to the *MW* and *kcat* data, which must be collected from literature and may not be available for all genes in the GEM.

    (MATLAB)

```
>tbl_ecoli = fillmissing(tbl_ecoli, "constant",

nanmedian(tbl{:, feat_names}), 'DataVariables', feat_names);
```

    b. Scale *geneKO* and *maxATPafterKO* by the wild-type growth rate.

    (MATLAB)

```
no_deletion_solution = optimizeCbModel(ecoli_GEM,[],'one');

wt_gr = no_deletion_solution.f;

tbl_ecoli{:,["geneKO","maxATPafterKO"]} = tbl_ecoli{:, ["geneKO", "maxATPafterKO"]}/
wt_gr;
```

    c. Limit *Vmax* and *Vmin* to values between -100 and 100.

    Note: This threshold is applied to reduce the impact of extremely large fluxes, which can indicate unconstrained reactions.

    (MATLAB)

```
Vlim = 100;

tbl_ecoli.Vmax(tbl_ecoli.Vmax > Vlim) = Vlim;

tbl_ecoli.Vmax(tbl_ecoli.Vmax < -Vlim) = -Vlim;

tbl_ecoli.Vmin(tbl_ecoli.Vmin > Vlim) = Vlim;

tbl_ecoli.Vmin(tbl_ecoli.Vmin < -Vlim) = -Vlim;
```

    d. Scale *kcat* and *pFBAflux* by their mean values.

    Note: We found that this method performed best for these features in terms of eliminating organism-specific signatures in the data distribution.

    (MATLAB)

```
>tbl_ecoli.kcat = tbl_ecoli.kcat / mean(tbl_ecoli.kcat);

>tbl_ecoli.PFBAflux = tbl_ecoli.PFBAflux / mean(tbl_ecoli.PFBAflux);
```

e. *reversible* is a binary value, so does not need scaling.
(MATLAB)

```
>tbl_ecoli.reversible = categorical(tbl_ecoli.reversible);
```

f. All features besides *kcat*, *pFBAflux* and *reversible* were normalized to values between 0 and 1.

Note: This step is not required for XGBoost, which is decision tree-based, however this step can be performed if the user is interested in trying different types of machine-learning algorithms which perform better with feature scaling.

(MATLAB)

```
num_vars = tbl_ecoli(:,vartype("numeric")).Properties.VariableNames;

norm_vars=num_vars;

norm_vars(ismember(norm_vars,["kcat","PFBAflux"])) = []

tbl_ecoli(:,norm_vars) = normalize(tbl_ecoli(:,norm_vars), 'range');
```

13. If multiple datasets or organism types are to be analyzed together, they should be combined at this stage. The datasets can simply be vertically concatenated.

(MATLAB)

```
>tbl_carom = [tbl_ecoli; tbl_yeast];
```

a. Save data to file.

(MATLAB)

```
>writetable(tbl_carom,"caromDataset.csv")
```

Running CAROM-ML in MATLAB

Timing: ~15 min

The CAROM-ML code was written in Python due to the machine-learning tools available in this language. However, if the user wishes to complete the entire project in MATLAB for simplicity reasons or unfamiliarity with Python, they can reference the below code for training a PTM-classification model.

14. Create a training and test set using the *E. coli* feature matrix that we created.

(MATLAB)

```
c = cvpartition(tbl_ecoli{:,'PTM'},'Holdout',0.2,'Stratify',true);

idxTrain = training(c);

tblTrain = tbl_ecoli(idxTrain,:);

idxVal = test(c);

tblVal = tbl_ecoli(idxVal,:);
```

15. Use the *fitcensemble* function with the "Bag" method to train a random forest model.

Note: For the CAROM-ML Python code, XGBoost is used for the primary machine-learning model, however this algorithm is not available in MATLAB.

(MATLAB)

```
t = templateTree('MaxNumSplits',5,...
  'NumVariablesToSample','all',...
  'PredictorSelection','interaction-curvature',...
  'Surrogate','on');
model = fitcensemble(tblTrain(:,feat_names),tblTrain{:,'PTM'},...
  'Method','Bag', 'Learners',t, 'NumLearningCycles',250);
```

16. Make predictions on the validation set using the trained model and evaluate the model performance by displaying the resulting confusion matrix.

(MATLAB)

```
> y_pred = predict(model, tblVal(:,feat_names));
> confusionmat(tblVal{:,'PTM'}, y_pred)
```

17. Plot the feature importance scores for the train model. For more information on how MATLAB calculates predictor importance, see the documentation page at this link.

(MATLAB)

```
[impGain, predAssociation] = predictorImportance(model);
figure()
bar(impGain);
title('Predictor Importance Estimates');
ylabel('Estimates');
xlabel('Predictors');
h = gca;
h.XTickLabel = model.PredictorNames;
h.XTickLabelRotation = 45;
h.TickLabelInterpreter = 'none';
```

Running CAROM-ML in Python

Timing: ~30 min

The following instructions detail how to load the previously formatted data into Python and use it as an input for the various CAROM-ML functions. In addition to training an XGBoost model for classifying gene-reaction pairs by PTM labels, these steps will demonstrate how to explain the model and its predictions.

18. Open your Python editor.

    Note: For the Python editor, we recommend using Spyder, an integrated development environment (IDE) that is included with Anaconda and allows you to run code interactively. By opening Spyder in the conda environment that was setup in the previous steps, all required Python libraries should already be installed and ready for use. For help on opening Spyder in a specific environment, see the "Frequently Asked Questions" page from Spyder's documentation.

19. Load the feature matrix and the CAROM-ML functions (found in *carom.py*) into your Python editor of choice.

    Note: Ensure that your working directory is set to your project folder (e.g., *Desktop/project_folder*) and that the conda environment has been activated (see troubleshooting Problems 1 and 2).

(Python)

```
# -*- coding: utf-8 -*-

"""

CAROM-ML script

"""

import pandas as pd

import carom

# load the CAROM data

df_carom = pd.read_csv("caromDataset.csv")
```

    a. You can inspect the first few rows of your data by running the following command:

(Python)

```
>df_carom.head()
```

If using Spyder, you can instead open the *df_carom* object in the Variable Workspace. The dataset should look similar to the example data shown in Table 2.

20. Train the CAROM-ML model using the train_model function.

    Note: This function trains an XGBoost model using cross-validation. Within each fold, hyperparameter tuning is performed and the model's performance on the test fold is recorded. By looking at the model's performance across all five folds, we gain a better idea of its robustness. By default, the train_model function performs 5-fold cross-validation. The number of folds may be altered using the num_folds argument, however the use should be wary of using

Table 2. Several rows from an example CAROM-ML input dataset

Genes	Reactions	geneKO	maxATP	. . .	Closeness	Pagerank	Kcat	MW	PTM
b0002	ASPK	0.921	1	. . .	0.538	0.239	1.004	0.537	0
b0002	HSDy	0.921	0	. . .	0.793	0.638	1.004	0.5376	0
b0003	HSK	0	1	. . .	0.827	0.699	0.527	0.275	1

The dataset should contain two columns containing the gene and reaction IDs, the 13 CAROM features and an array of PTM labels.

too many folds on highly imbalanced datasets, thereby leaving too few regulated gene-reaction pairs in each fold. After cross-validation, a final model is fitted to the entire dataset using the hyperparameters from the best performing fold.

a. First, split the data into a training and validation set, so that there is data we can later test on which the model has not been exposed to. If you already have a validation dataset to test on, you can skip this step.

(Python)

```
from sklearn.model_selection import train_test_split

% define the feature names

feature_names = df_carom.columns[3:16]

% create X and y variables, then split into training-test

X = df_carom[feature_names]

y = df_carom['PTM']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123,
stratify=y)
```

b. Use the X (independent features) and y (class labels) training data as inputs into the train_-model function.

(Python)

```
[xgb_model, scores] = carom.train_model(X=X_train, y=y_train, class_names=["Acetyl","Un-
known","Phos"], num_iter=5, condition="TestRun", fig_path="./figures/training")
```

Note: For any of the CAROM Python functions, running the help function will provide details on the required and optional arguments:

(Python)

```
>help(carom.train_model)
```

c. Here we will highlight the possible inputs for the *imbalance* argument, which dictates how the model training handles class imbalance.
   i. Dictionary OR "auto": the Adaptive Synthetic (ADASYN) algorithm is implemented. ADASYN is a method used for over-sampling the minority classes, but instead of replicating existing observations (the feature values for gene-reaction pairs), entirely new samples are interpolated (He et al., 2008). The user can enter a dictionary with the classes and the corresponding number of samples they want for each class. For example, if the

unknown class contains 500 samples, but the phosphorylation and acetylation class have only a fraction of this, the user may wish to generate twice as many samples for the two classes of interest:

```
>imbalance = {0:1000, 1:500, 2:1000}
```

Note that due to the way the class labels are encoded, the first class in the dictionary should always be set to zero. To generate an even number of samples for all three classes (e.g., 500 each), the user can simply enter "auto".

ii. "undersample": random samples from the larger classes in the dataset are removed until all classes have the same number of samples.

iii. "balanced": the inverse proportion of classes are used to assign a balanced set of class weights. Classes with less samples will be given higher weights, meaning that the machine-learning algorithm will more heavily penalize misclassifying those points.

iv. "none" (default): no adjustments are made.

d. The two outputs from this function are the trained XGBoost model and a data frame containing the cross-validation classification scores. Additionally, three figures should be saved to the folder specified in the fig_path argument: a confusion matrix with the cross-validation results, a bar graph showing the cross-validation scores (the error bars represent standard deviation) and the XGBoost feature importance plot (Figure 2). Note that the feature importance plot here is intended to assist with model explanation, as opposed to feature selection. The feature importance scores demonstrate how all features contribute to the model predictions with different degrees of importance, which is also highlighted in the ensuing Shapley analysis.

21. Make predictions on the validation dataset using the trained model and the make_predictions function.

Note: This function returns the model's classification scores on the validation set, as well as the list of predicted class labels. Additionally, a CSV file is saved for each class indicating the model's performance for each gene-reaction pair (true/false positive, true/false negative). If the user enables the *plot* argument within the function, a bar graph with the classification scores is generated, as well as box plots for each numeric feature in each class. Example plots are shown in Figure 3, including the confusion matrix (A), grouped box plots (B) and classification scores bar plot (C).

(Python)

```
# get the gene-reactions pairs in the test set

ix = X_test.index

test_genes = df_carom.loc[ix,['genes','rxns']]

# make predictions on test set

[df_scores, ypred] = carom.make_predictions(mdl= xgb_model,

          X_test = X_test, y_test = y_test,

          class_names = ["Phos","Unknown","Acetyl"],

          gene_reactions = test_genes,

          plot = True)
```

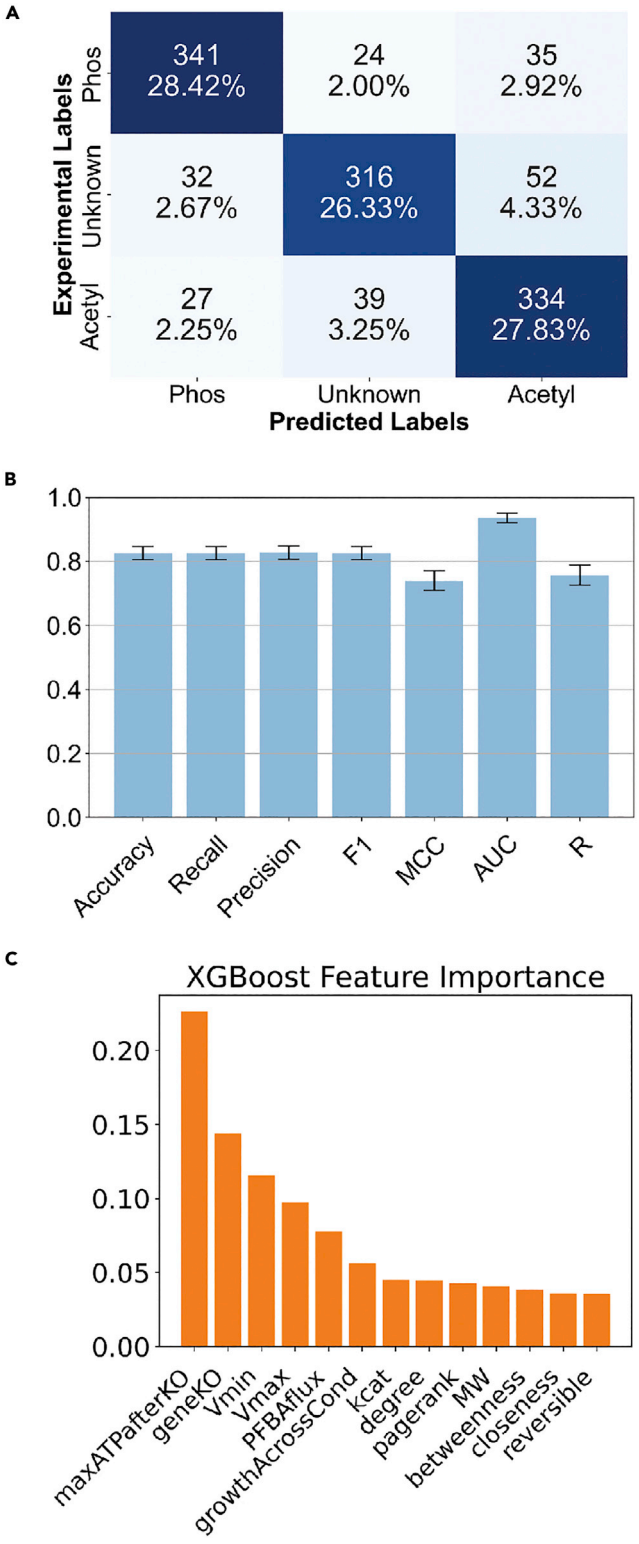

A
341
28.42%
24
2.00%
35
2.92%
32
2.67%
316
26.33%
52
4.33%
27
2.25%
39
3.25%
334
27.83%
Experimental Labels
Phos
Unknown
Acetyl
Phos
Unknown
Acetyl
Predicted Labels
B
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
Recall
Precision
F1
MCC
AUC
R
C
XGBoost Feature Importance
0.20
0.15
0.10
0.05
0.00
maxATPafterKO
geneKO
Vmin
Vmax
PFBAflux
growthAcrossCond
kcat
degree
pagerank
MW
betweenness
closeness
reversible

Figure 2. Plots generated from the carom.train_model function

(A) Confusion matrix showing the cumulative classification results from the cross-validation training of the XGBoost model.

(B) Various classification scoring metrics are plotted for the cross-validation results. The error bars shown represent the 95% confidence interval for each metric.

(C) The feature importance scores from the XGBoost model. By default, XGBoost feature importance is measured in gain, which is the improvement in accuracy that a feature adds to the branches it is on (click here for more information).

22. If you do not know the true labels of the test set, or if you are simply interested in getting a list of predicted labels for the test set, use the following command:

(Python)

```
>ypred = xgb_model.predict(X_test)
```

23. Use the shapley function to explain and interpret the trained model.

> ***Note:*** This function is a wrapper for several functions from the SHAP library. While Shapley values can potentially be used for feature selection or reduction, the primary purpose of this function is to aid in model interpretation.

The inputs include the XGBoost model and the feature matrix for the observations that we want to explain (e.g., *X_test*).

(Python)

```
[shap_explainer, shap_values] = carom.shapley(
  xgbModel = xgb_model,
  X = X_test[feature_names],
  condition = "TestInstall",
  fig_path = "./figures/shap")
```

The function's outputs include the SHAP explainer object and the Shapley values. The *shap_values* variable should be a matrix of *N* observations x *M* features. The explainer object may be useful if the user wishes to perform their own additional analyses with the SHAP package. Examples of the output plots are shown in Figure 4. It is recommended to review SHAP documentation page for interpreting these figures (https://shap.readthedocs.io/en/stable/index.html).

24. Analyze specific genes of interest using the select_genes function, as shown in Figure 5.

> ***Note:*** The required inputs include the feature matrix and PTM labels from the background dataset (e.g., *X_test, y_test*), the gene-reactions pairs associated with that data, a list of genes of interest, the trained XGBoost model and the SHAP explainer from the above step. A multi-output decision plot will be generated for each reaction linked to the genes of interest.

a. In the below example, we pick two gene-reaction pairs to analyze: the first instance of phosphorylation and acetylation in the test dataset. In order to calculate the Shapley values for these two observations, we have to pass several pieces of data to the function, including the test data, the entire set of gene-reaction pairs, the trained XGBoost model and the Shapley explainer from the previous section.

(Python)

```
# get first Phos gene and first Acetyl gene

phos_gene = test_genes.genes[y_test==-1].iloc[0]

acetyl_gene = test_genes.genes[y_test==1].iloc[0]

select_genes = [phos_gene, acetyl_gene]

carom.select_genes(X = X_test, y = y_test,

       all_geneRxns = test_genes,

       select_genes = select_genes,

       condition = "my_selecet_genes",

       class_names = ["Phos", "Unknown", "Acetyl"],

       model = xgb_model,

       explainer = shap_explainer)
```

25. Build a single decision tree model using the decision_tree function, which provides another method for deconstructing the model's predictions and measuring feature importance.

    Note: This function is a wrapper for the decision tree classifier and hyperparameter tuning from scikit learn. The outputs of this function include a separate decision tree model trained for each maximum depth that the user inputs, along with a saved PNG file for visualizing the tree (Figure 6). For example, if the user inputs depths=[2, 3], two models will be stored in the assigned variable and two images should be saved to the project folder.

(Python)

```
my_weights = [{0:2,1:1,2:2}, {0:5,1:1,2:5}]

my_trees = carom.decisionTree(X = df_carom[feature_names],

       y = df_carom["PTM"],

       class_names = ["Phos", "Unknown", "Acetyl"],

       imbalance=my_weights,

       condition = "weighted_dTree",

       make_viz=True,

       depths=[2],

       pruneLevel=5,

       random_seed=123)
```

    a. Class imbalance: similar to the train_model function, several options are available within the *imbalance* argument for adjusting how the model addresses class imbalance.
        i. The user may enter a list of dictionaries, each containing the class weights (see the example above). As described previously, typically the less represented classes are given higher weights, so that the model penalizes misclassifications of that class more severely. In this case, a separate model is trained using cross-validation and hyperparameter tuning for each set of weights within the list. This allows the user to experiment with different

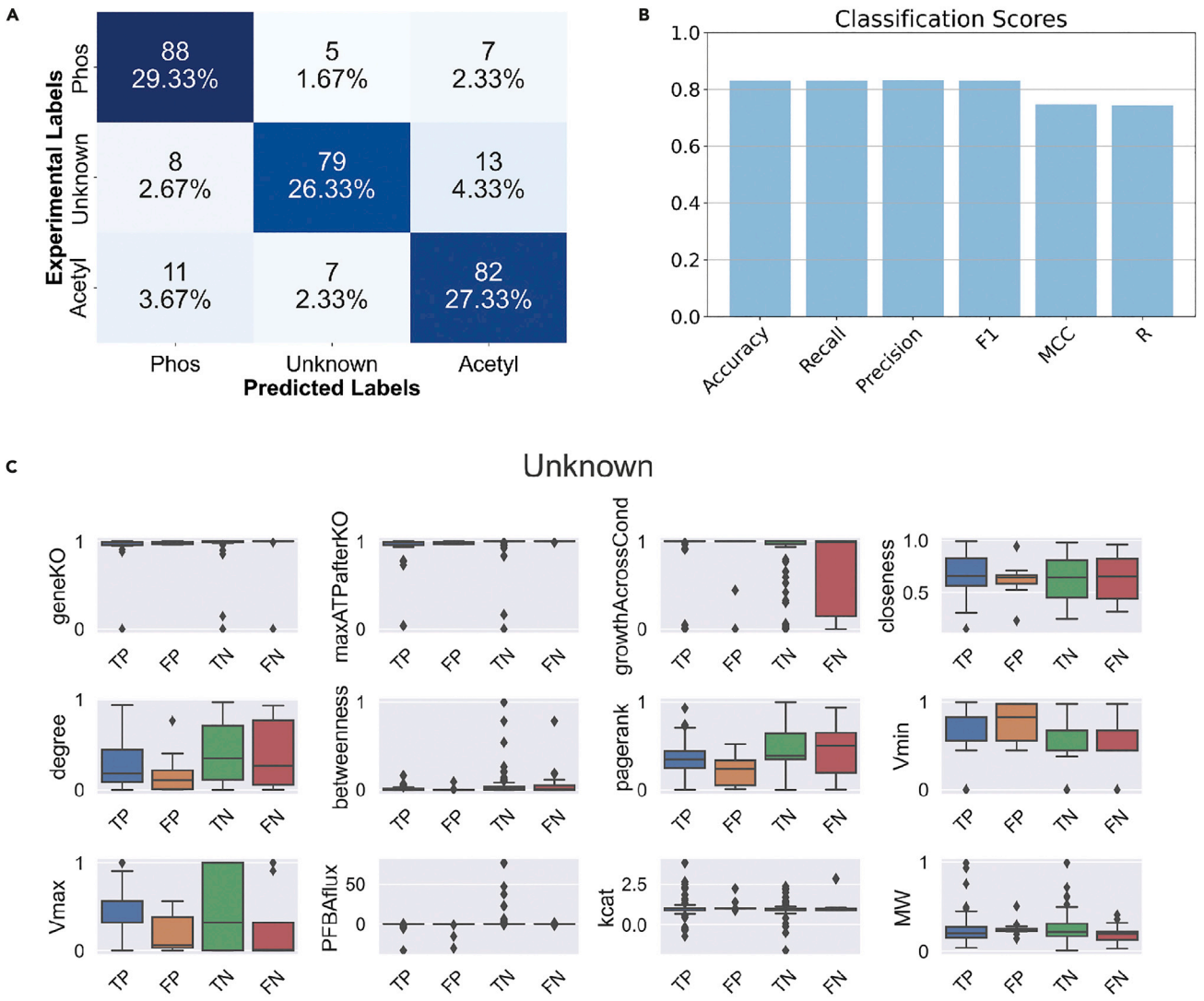


Figure 3. The `carom.make_predictions` function is used to test the trained CAROM-ML model on a validation dataset where the true PTM labels are already known

Several figures are generated by default which help visualize the model's performance.

(A) Confusion matrix showing the true labels on the y-axis and the predicted labels on the x-axis.

(B) Various classification metrics showing the model's performance on the test set.

(C) Boxplots for each numerical feature and for each class, showing the feature distribution grouped by true positives, false positives, true negatives and false negatives.

sets of weights. The model with the best cross-validation score is then selected as the final model.

ii. "adasyn": Uses the ADASYN oversampling method described in the train_model section, with all classes set to the same size.

iii. "balanced": the inverse proportion of classes are used to assign a balanced set of class weights.

iv. "none": no adjustments are made to the model training.

b. Pruning – the *pruneLevel* argument is used to remove any leaves on the decision tree with less observations than the specified integer.

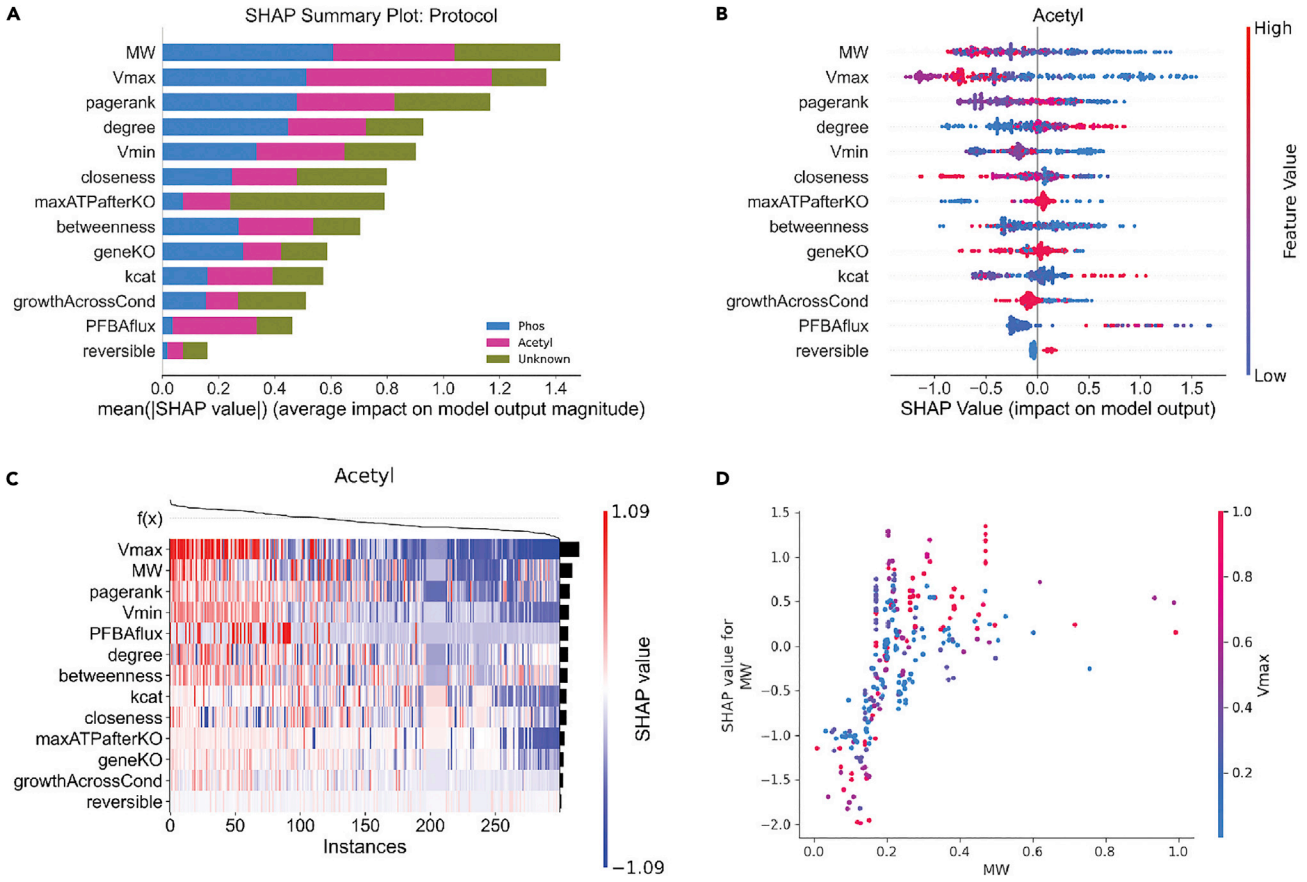


Figure 4. The `carom.shapley` function outputs several figures from the SHAP package, all of which help explain the model's predictions

(A) The multi-class summary plot shows the feature importance according to the Shapley values across all three classes.

(B) A separate Shapley summary plot is generated for each class. The features are sorted from top to bottom by average absolute Shapley value for the specific class. Unlike the multi-class plot, the single class plot shows how each feature is correlated with the model's output.

(C) A Shapley value heatmap is created for each class. The model's output for the respective class is shown on the top x-axis in log odds, the features are ordered on the y-axis by importance, and the observations are clustered according to f(x).

(D) For each class, a Shapley dependence plot is generated for the top three most importance features (according to that class' summary plot). The feature's values are on the x-axis and the feature's Shapley values on the y-axis. The data points are colored by the values of the feature which most interacts with the main feature of the plot.

EXPECTED OUTCOMES

The primary outcome of this analysis is a better understanding of why metabolic enzymes are targeted by a certain PTM in the condition of interest. By following the CAROM-ML pipeline, the user is left with a trained machine-learning model which can be used to make predictions for gene-reaction pairs whose regulation is unknown. More importantly, the CAROM-ML tool contains several methods for explaining these predictions, which are based on a small set of thirteen features related to biochemical properties, metabolic activity, and network connectivity.

As demonstrated in the graphical abstract figure, the bulk of the work for the user falls within setting up and preprocessing the feature matrix in MATLAB. Six of features used in the machine-learning model are generated from flux balance analyses. It is critical that the GEM being used for these calculations is reflective of the condition of interest. This may require adjusting the GEM using condition-specific transcriptomics. Once the entire feature matrix has been created and PTM labels have been assigned to the gene-reaction pairs, the CAROM-ML functions in Python make it easy to begin creating and interpreting a ML model based on the data. The primary analyses include training a

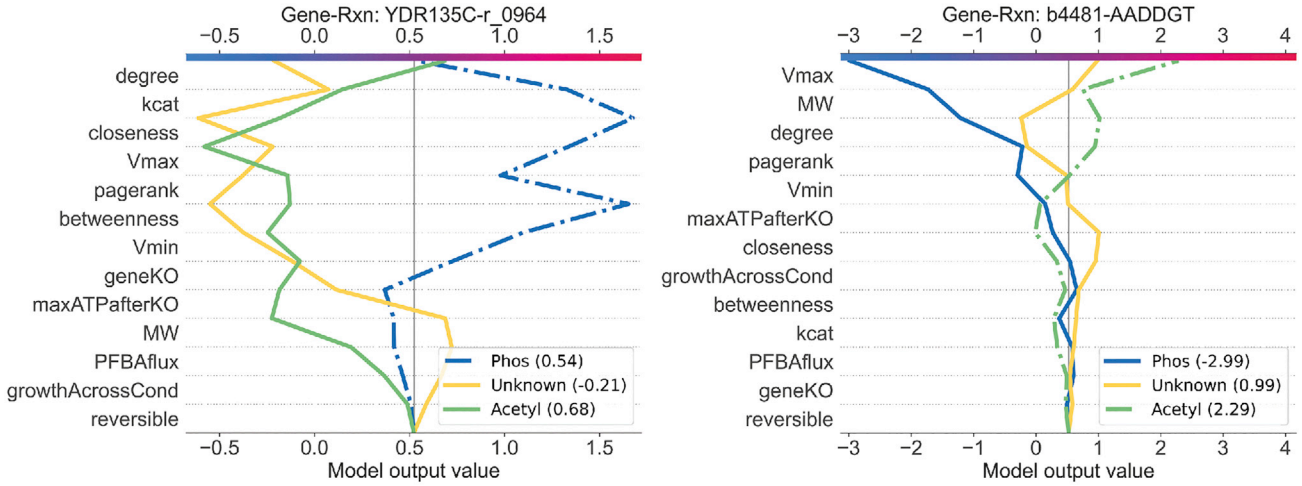


Figure 5. The carom.select_genes function outputs Shapley decision plots for each gene-reaction pair associated with the input genes

The features are ordered by descending importance for the specific observation. The gray line intersects the x-axis at the expected value, which is essentially the model's average prediction for that class in log odds. Moving up the plot from this mark on the x-axis, we can see how the model's output for each class is changed as features are added. The final outputs are where the colored lines intersect the top x-axis, meaning that the line farthest to the right corresponds to the model's prediction. The dashed line represents the true class.

XGBoost model and using Shapley values to explain both the entire model and specific gene-reaction pairs. These results can help uncover which enzyme properties play a role in dictating why a PTM is or is not active in a condition. While this approach has currently only been applied to two types of regulation (phosphorylation and acetylation), in the future the types of PTMs could be expanded upon. Ultimately the goal is for CAROM-ML to guide metabolic engineering and drug discovery efforts by informing researchers which regulators to expect in various parts of the metabolic network.

LIMITATIONS

The main limitations of this study stem from the use of GEMs. Because roughly half of the features used in the machine-learning tool are generated using FBA methods, any inaccuracies within the GEM may have significant effects on the classification results. However, previous studies have shown that fluxes generated from FBA closely resemble their corresponding experimental values. We have also shown that CAROM-ML's PTM predictions closely resemble experimental regulation data.

Another limitation is the availability of training data. When applying a previously trained CAROM model to a new condition, only the condition-specific parameters are needed to update the flux data and generate PTM labels. However, when training a new model, data availability may be another limiting factor. In addition to needing a GEM that represents the organism of interest, the user must be able to gather the enzyme-specific properties (*MW*, *kcat*), as well as the phosphoproteomics and acetylomics levels. If data is not available for all genes present in the GEM, the user must decide between filling in the missing data or removing those genes. In the primary study, missing feature data was replaced by the median value, while missing target labels were assigned to the "unknown" class. If there was zero literature evidence regarding the acetylation or phosphorylation of the missing gene, then it was removed entirely.

TROUBLESHOOTING

Problem 1

There are several issues that may cause an error when trying to import Python packages into your Python editor in Step 18, such as shown here:

```
>ModuleNotFoundError: No module named 'pandas'
```

Potential solution

- The first thing to check is that the conda environment, *conda-env*, exists and has been properly setup using the *environment.yml* file. To do this, open the command line and enter the following command in order to view a list of all of your conda environments:

(Windows/MacOS)

```
>conda env list
```

This command should produce a result similar to the line below if you have only the provided environment installed:

```
>C:\Users\username\Desktop\project_folder\conda-env
```

- If this looks correct, the next step is to make sure that your editor has been opened within the environment. We will assume that Spyder is being used as the editor, as suggested. After opening Anaconda Navigator, it is important to switch the working environment. It may be necessary to select the correct environment from the dropdown menu, as shown in the red rectangle in Figure 7.

Once the correct environment has been selected, launch Spyder and the necessary libraries should be ready to be imported.

Alternatively, Spyder can be opened in the CAROM-ML environment via the terminal (or using Anaconda Prompt on Windows):

(Windows/MacOS)

```
cd project_folder

>conda activate conda-env

>spyder
```

Problem 2

Similar to problem 1, you may encounter the following error in Step 18 when trying to read in data into Spyder.

```
FileNotFoundError: [Errno 2] No such file or directory: 'caromDataset.csv'
```

Potential solution

- Ensure the Spyder working directory is set to the project folder. The working directory can be changed using the dropdown menu in the top right of the console, as shown In Figure 8.
- If the datasets are not saved in the main folder, ensure to either adjust the relative paths or to use absolute file paths (e.g., *'C:\Users\username\Desktop\project_folder\caromDataset.csv'*).

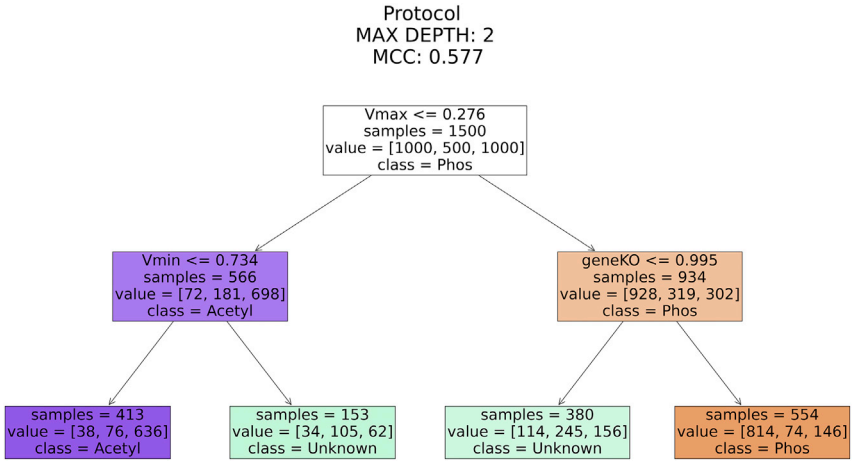


Figure 6. The carom.decision_tree function is another tool for understanding which features are instrumental in differentiating between PTM classes

The function outputs a separate decision tree for each maximum depth value that is specified. The colors indicate the majority class for each leaf, while the strength of the colors represents how dominant the major class is.

Problem 3

The imbalanced nature of the omics data can make it difficult for models to predict instances of regulation. When training the model in Step 19, the model may be biased by the much higher number of "unknown" gene-reaction pairs, compared to the much lower number of phosphorylated or acetylated pairs.

Potential solution

- XGBoost handles class imbalance well by default. However, you may still find that your model has poor recall on a test set or is predicting a lower-than-expected number of regulated gene-reaction

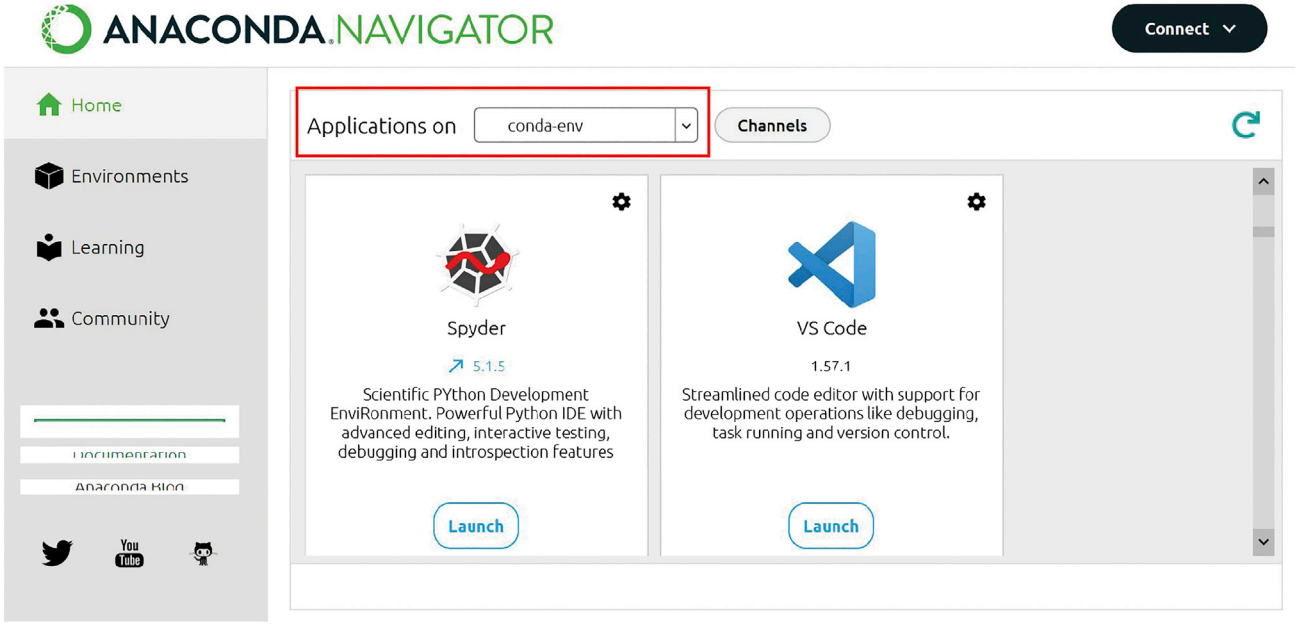


Figure 7. After opening Anaconda Navigator, ensure the "Applications on" dropdown menu is set to the correct environment

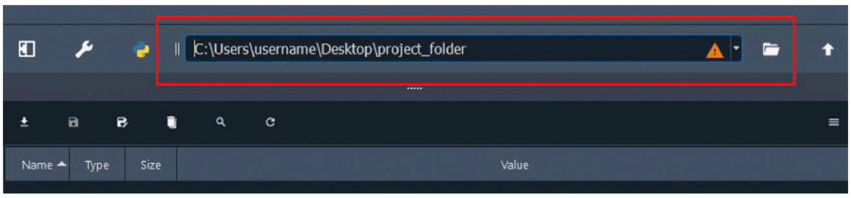


Figure 8. If Spyder is not set to the correct working directory, any relative file paths will not work for reading in data or Python libraries

pairs. The *imbalance* argument for the train_model can be used to adjust how the model's training takes the minority classes into account. Refer to section 2c of running CAROM-ML in Python for details on how to incorporate either oversampling, undersampling, or class weights into the model training.

Problem 4

The XGBoost model's performance in the cross-validation training is much different than the model's performance on test or validation sets in Step 20.

Potential solution

- This may be due to the model being over- or under-fitted. If that is the case, try adjusting the optional arguments of the train_model function. For example, for if you suspect the model is over-fitted, ensure that *depth* is set to "shallow" in order to train less-complex trees within the model, or try reducing the value of *num_iter*, which controls the number of hyperparameter tuning searches.
- If the XGBoost training does not seem to be the issue, compare the distribution of the features in the training versus validation data. If they are not similar, additional preprocessing may be required.

Problem 5

The COBRA toolbox functions in Step 5 are not working or is producing unexpected results regarding the flux balance-related features.

Potential solution

- First, ensure that your optimization solver is compatible with COBRA.
- Next, ensure that you use the following command to load the GEM into MATLAB, as other methods can cause issues.

(MATLAB)

```
>readCbModel(GEM)
```

- Run the following commands to identify potential issues with the GEM's structure fields and missing biomass precursors, respectively.

(MATLAB)

```
>verifyModel(GEM)

>biomassPrecursorCheck(GEM)
```

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Sriram Chandrasekaran (csriram@umich.edu).

Materials availability

No materials were generated in this study.

Data and code availability

- All data have been deposited at Synapse and are publicly available as of the date of publication at https://www.synapse.org/CAROM. DOIs are listed in the key resources table.
- All original code has been deposited at Synapse and is publicly available as of the date of publication. DOIs are listed in the key resources table.
- Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

ACKNOWLEDGMENTS

This work was supported by faculty start-up funds from the University of Michigan, Camille and Henry Dreyfus Foundation, and R35 GM13779501 from NIH to S.C.

AUTHOR CONTRIBUTIONS

S.C. conceived the study; S.C. and K.S. designed and performed research; and S.C., N.R., and K.S. wrote the protocol.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Campit, S., and Chandrasekaran, S. (2020). Inferring metabolic flux from time-course metabolomics. Methods Mol. Biol. *2088*, 299–313.

Chandrasekaran, S., Zhang, J., Sun, Z., Zhang, L., Ross, C.A., Huang, Y.C., Asara, J.M., Li, H., Daley, G.Q., and Collins, J.J. (2017). Comprehensive mapping of pluripotent stem cell metabolism using dynamic genome-scale network modeling. Cell Rep. *21*, 2965–2977. https://doi.org/10.1016/j.celrep.2017.07.048.

Chen, T., and Guestrin, C. (2016). XGBoost: a scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Association for Computing Machinery (KDD'16)), pp. 785–794.

Chung, C.H., Lin, D.W., Eames, A., and Chandrasekaran, S. (2021). Next-generation genome-scale metabolic modeling through integration of regulatory mechanisms. Metabolites *11*, 606. https://doi.org/10.3390/metabo11090606.

He, H., Bai, Y., Garcia, E.A., and Li, S. (2008). ADASYN: adaptive synthetic sampling approach for imbalanced learning. In 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence) (IEEE), pp. 1322–1328.

Heirendt, L., Arreckx, S., Pfau, T., Mendoza, S.N., Richelle, A., Heinken, A., Haraldsdóttir, H.S., Wachowiak, J., Keating, S.M., Vlasov, V., et al. (2019). Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0. Nat. Protoc. *14*, 639–702.

Houser, J.R., Barnhart, C., Boutz, D.R., Carroll, S.M., Dasgupta, A., Michener, J.K., Needham, B.D., Papoulas, O., Sridhara, V., Sydykova, D.K., et al. (2015). Controlled measurement and comparative analysis of cellular components in E. coli reveals broad regulatory changes in response to glucose starvation. PLoS Comput. Biol. *11*, e1004400. https://doi.org/10.1371/journal.pcbi.1004400.

Kori, Y., Sidoli, S., Yuan, Z.F., Lund, P.J., Zhao, X., and Garcia, B.A. (2017). Proteome-wide acetylation dynamics in human cells. Sci. Rep. *7*, 10296.

Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.I. (2020). From local explanations to global understanding with explainable AI for trees. Nat. Mach. Intell. *2*, 56–67.

Mahadevan, R., and Schilling, C.H. (2003). The effects of alternate optimal solutions in constraint-based genome-scale metabolic models. Metab. Eng. *5*, 264–276.

Murphy, J.P., Stepanova, E., Everley, R.A., Paulo, J.A., and Gygi, S.P. (2015). Comprehensive temporal protein dynamics during the diauxic shift in Saccharomyces cerevisiae. Mol. Cell. Proteomics *14*, 2454–2465.

Oliveira, A.P., Ludwig, C., Picotti, P., Kogadeeva, M., Aebersold, R., and Sauer, U. (2012). Regulation of yeast central metabolism by enzyme phosphorylation. Mol. Syst. Biol. *8*, 623.

Olsen, J.V., Vermeulen, M., Santamaria, A., Kumar, C., Miller, M.L., Jensen, L.J., Gnad, F., Cox, J., Jensen, T.S., Nigg, E.A., et al. (2010). Quantitative phosphoproteomics reveals widespread full phosphorylation site occupancy during mitosis. Sci. Signal. *3*, ra3.

Orth, J.D., Conrad, T.M., Na, J., Lerman, J.A., Nam, H., Feist, A.M., and Palsson, B.Ø. (2011). A comprehensive genome-scale reconstruction of Escherichia coli metabolism–2011. Mol. Syst. Biol. *7*, 535.

Orth, J.D., Thiele, I., and Palsson, B.Ø. (2010). What is flux balance analysis? Nat. Biotechnol. *28*, 245–248.

Shen, F., Cheek, C., and Chandrasekaran, S. (2019). Dynamic network modeling of stem cell metabolism. Methods Mol. Biol. *1975*, 305–320.

Smith, K., Shen, F., Lee, H.J., and Chandrasekaran, S. (2022). Metabolic signatures of regulation by phosphorylation and acetylation. iScience *25*, 103730.

Soares, N.C., Spät, P., Krug, K., and Macek, B. (2013). Global dynamics of the Escherichia coli proteome and phosphoproteome during growth in minimal medium. J. Proteome Res. *12*, 2611–2621.

Treu, L., Campanaro, S., Nadai, C., Toniolo, C., Nardi, T., Giacomini, A., Valle, G., Blondin, B., and

Corich, V. (2014). Oxidative stress response and nitrogen utilization are strongly variable in Saccharomyces cerevisiae wine strains with different fermentation performances. Appl. Microbiol. Biotechnol. *98*, 4119–4135.

Weinert, B.T., Iesmantavicius, V., Wagner, S.A., Schölz, C., Gummesson, B., Beli, P., Nyström, T., and Choudhary, C. (2013). Acetyl-phosphate is a critical determinant of lysine acetylation in E. coli. Mol. Cell *51*, 265–272.

Weinert, B.T., Iesmantavicius, V., Moustafa, T., Schölz, C., Wagner, S.A., Magnes, C., Zechner, R., and Choudhary, C. (2014). Acetylation dynamics and stoichiometry in Saccharomyces cerevisiae. Mol. Syst. Biol. *10*, 716. https://doi.org/10.1002/msb.134766.