Interpretable Deep Learning

Interpretable vs Explainable

- After: explainable AI (XAI)
- Before: interpretable AI

After: Explainable AI

- Feature importance, partial dependency plots
- Shapley values
- Saliency maps, Integrated gradients, layerwise relevance propagation, etc.
- Global surrogate models
- Local surrogate models (e.g., LIME)
- Counterfactuals

Interpretable Machine Learning, C. Molnar (2020)

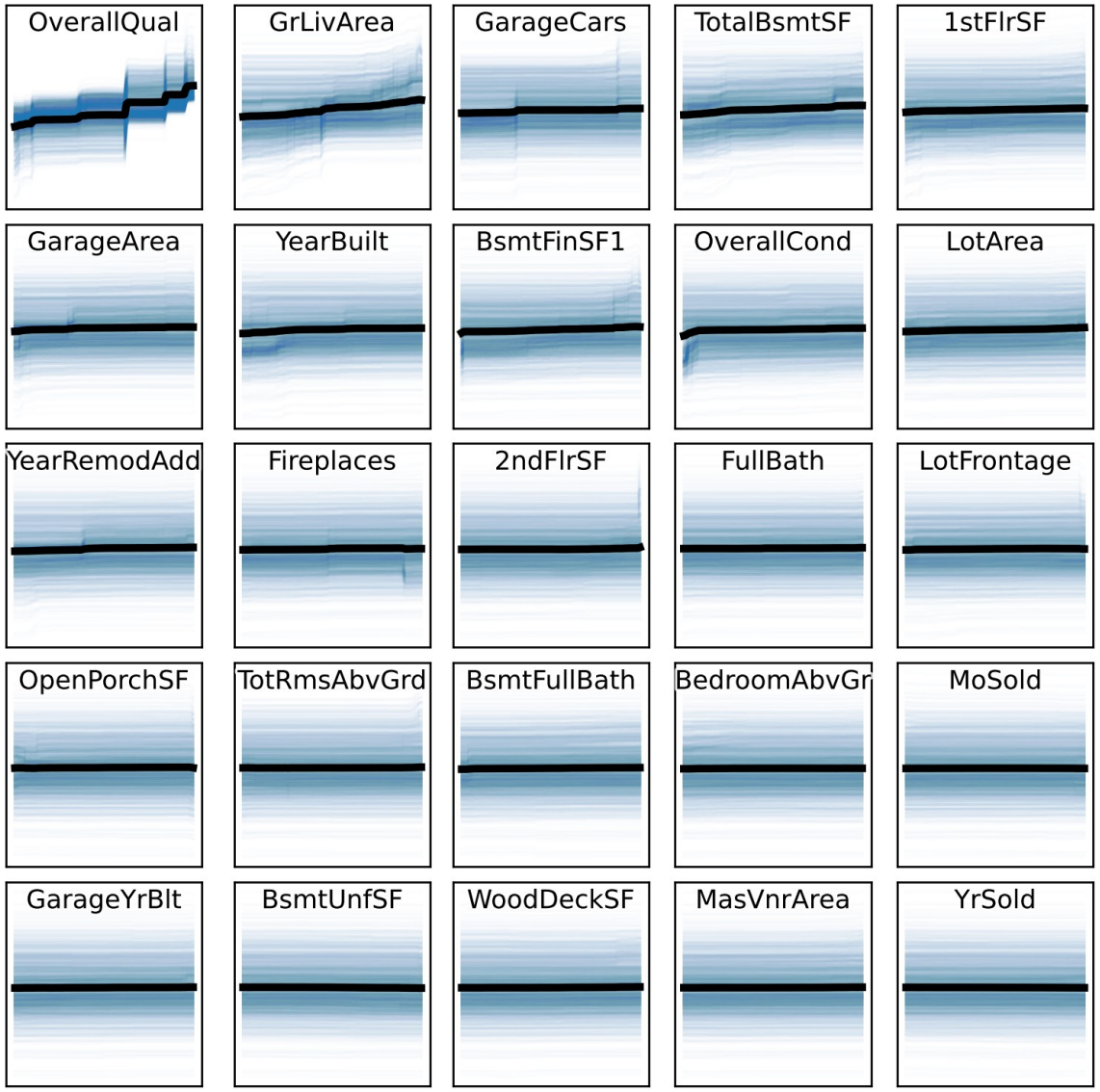
After: Explainable AI

- Feature importance, **partial dependency plots**
- Shapley values
- Saliency maps, Integrated gradients, layerwise relevance propagation, etc.
- Global surrogate models
- Local surrogate models (e.g., LIME)
- Counterfactuals

Interpretable Machine Learning, C. Molnar (2020)

Partial Dependency Plot

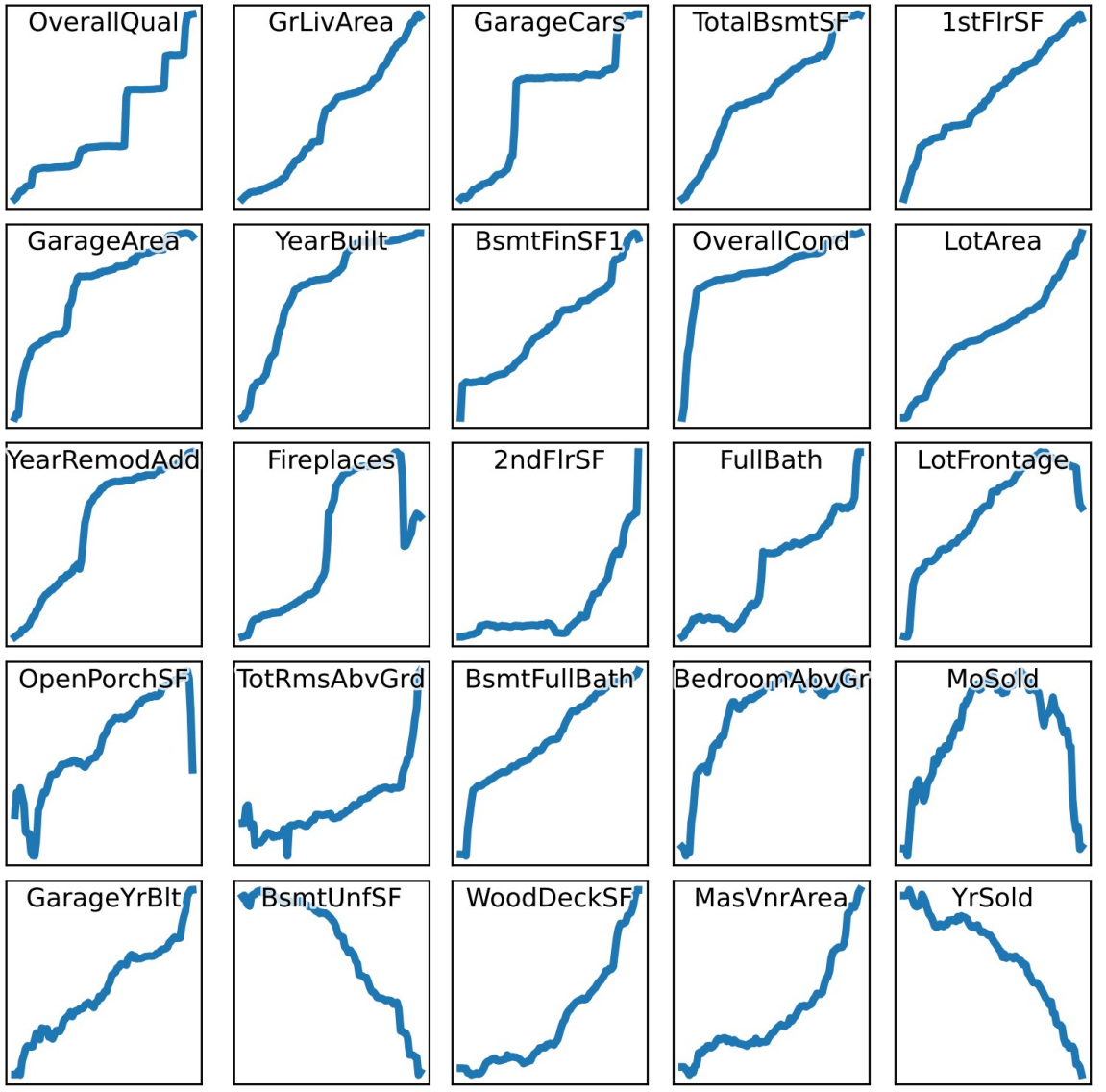
- For each observation i, how does the output change if we change variable j (while keeping the others constant)?
- Average of those curves



Interpretable Machine Learning, C. Molnar (2020)

Partial Dependency Plot

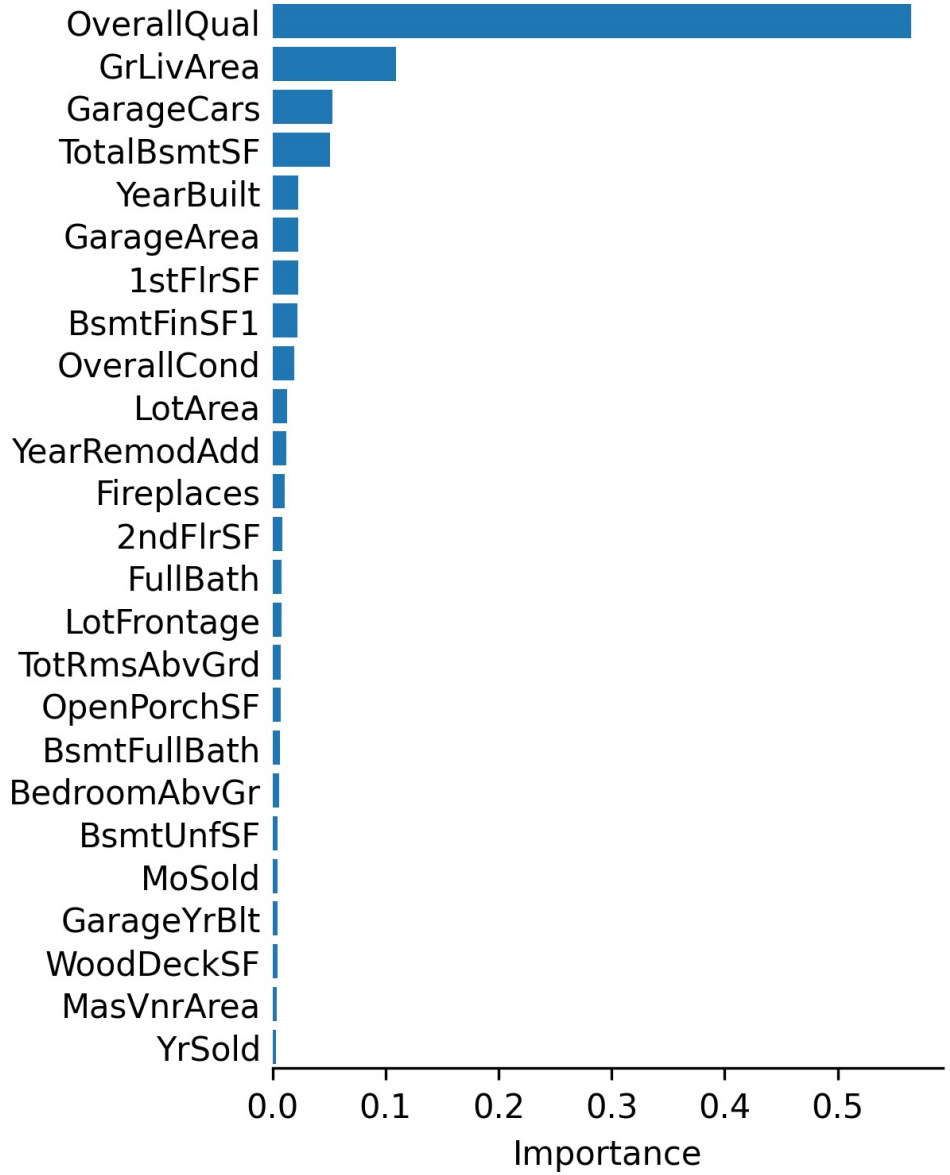
- For each observation i, how does the output change if we change variable j (while keeping the others constant)?
- Average of those curves



Interpretable Machine Learning, C. Molnar (2020)

Feature Importance

- How does the accuracy of the model decrease if we remove (or shuffle) one input variable?



What makes a model interpretable?

No rigorous definition:

- Sparsity
- Monotonicity
- Modularity
- Linearity

Designing inherently interpretable machine learning models, A. Sudjianto and A. Zhang (2021)

Before: Interpretable AI

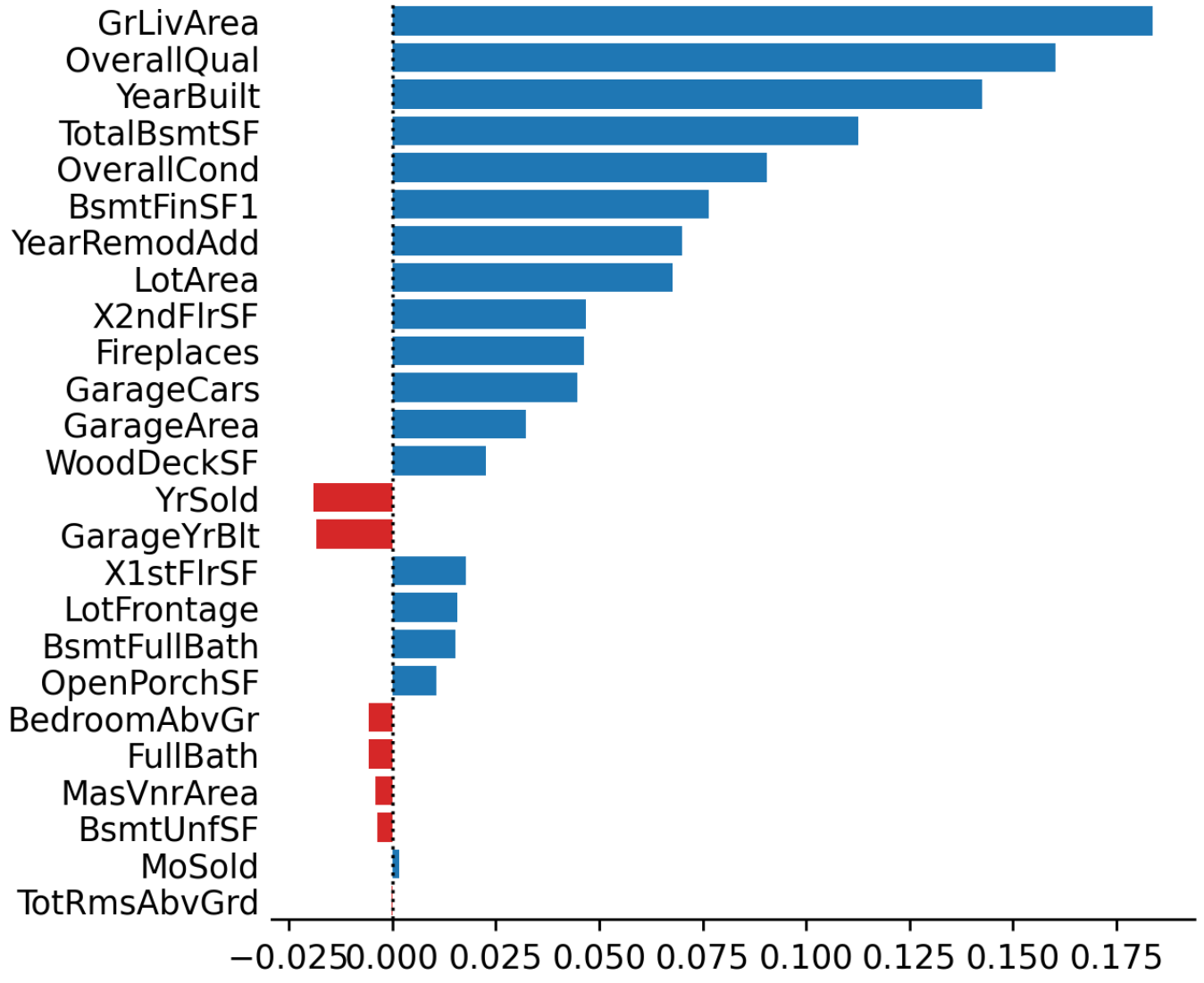
- Linear Models
  - Lasso: fused, group, graph sparsity
  - Sign constraints
  - SLIM
- ML models
  - Rule lists, decision trees
  - Gradient boosting
  - GAM, GA²M, AIM
- Deep learning models
  - Monotonicity: activation functions, gradient penalty
  - Sparsity
  - Varying coefficient models
  - Mixture of experts

Linear Models

Linear regression

$$\begin{aligned} \text{Find} \quad & w \in \mathbf{R}^k \\ \text{To minimize} \quad & \|Xw - y\|_2^2 \end{aligned}$$

```
# Linear regression
model = LinearRegression()
model.fit(X, y)
yhat = model.predict(X)
```



Lasso

$$\begin{aligned} &\text{Find} && w \in \mathbf{R}^k \\ &\text{To minimize} && \|Xw - y\|_2^2 + \lambda \|w\|_1 \end{aligned}$$

```
# Linear regression with L¹ penalty
model = sklearn.linear_model.Lasso(alpha=1)
model.fit(X, y)
yhat = model.predict(X)
```

Statistical Learning with Sparsity, T. Hastie et al. (2015)

Lasso

$$\begin{aligned} &\text{Find} && w \in \mathbf{R}^k \\ &\text{To minimize} && \|Xw - y\|_2^2 + \lambda \|w\|_1 \end{aligned}$$

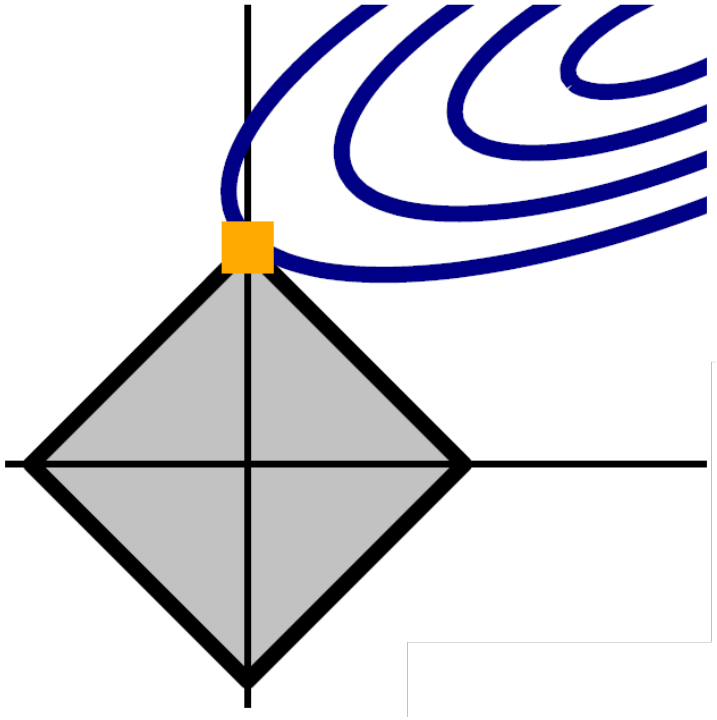
```
import cvxpy as cp
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - ( intercept + X.values @ w.T )
objective = cp.sum_squares( residuals ) / n + alpha * cp.norm1(w)
prob = cp.Problem( cp.Minimize(objective), [] )
result = prob.solve()
w = pd.Series( w.value, index = X.columns )
w[ np.abs(w) < 1e-6 ] = 0
```

Statistical Learning with Sparsity, T. Hastie et al. (2015)

Lasso: sparsity

$$\begin{array}{ll} \text{Find} & w \in \mathbf{R}^k \\ \text{To minimize} & \|Xw - y\|_2^2 + \lambda \|w\|_1 \end{array}$$

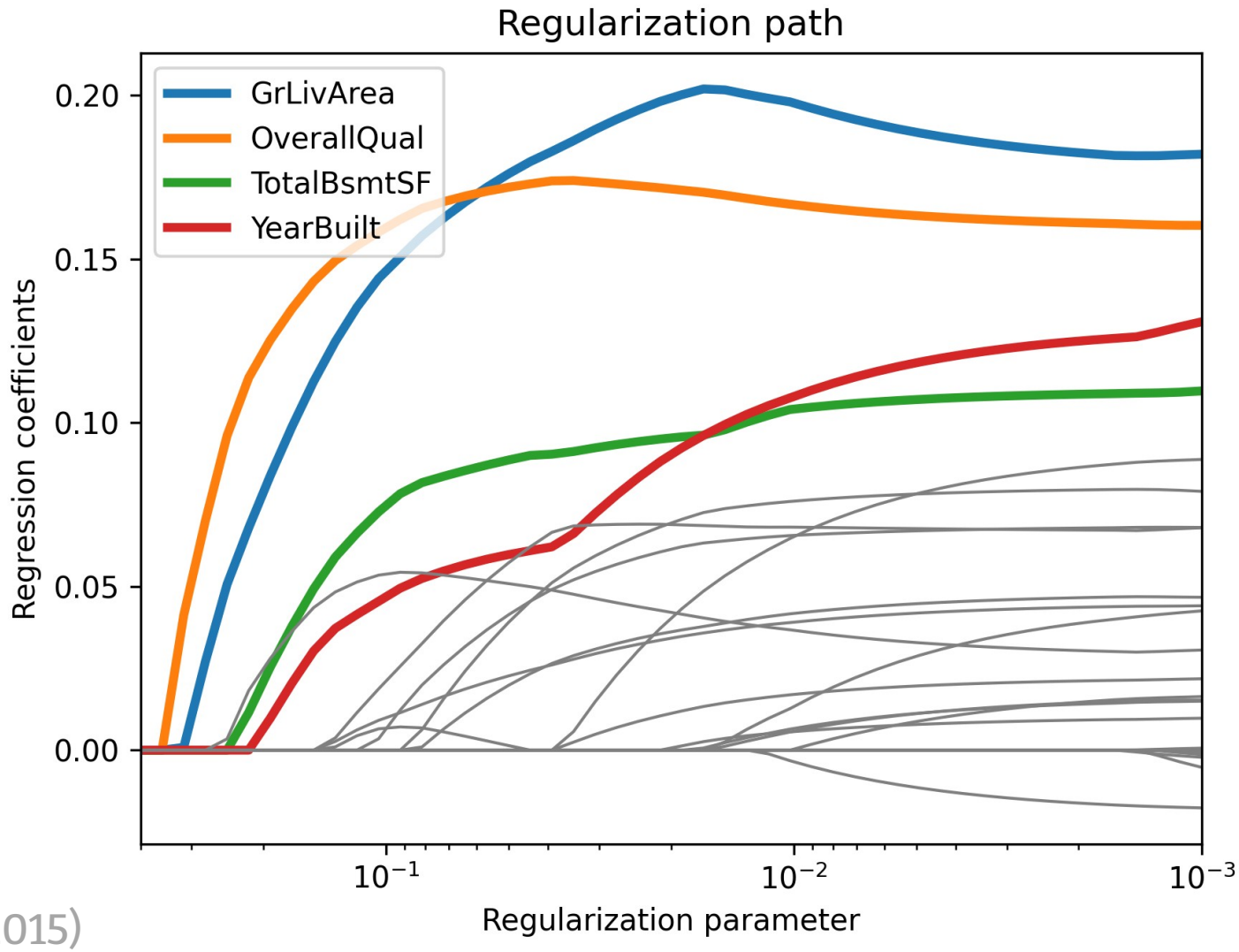
$$\begin{array}{ll} \text{Find} & w \in \mathbf{R}^k \\ \text{To minimize} & \|Xw - y\|_2^2 \\ \text{Such that} & \|w\|_1 \leqslant c \end{array}$$



Statistical Learning with Sparsity, T. Hastie et al. (2015)

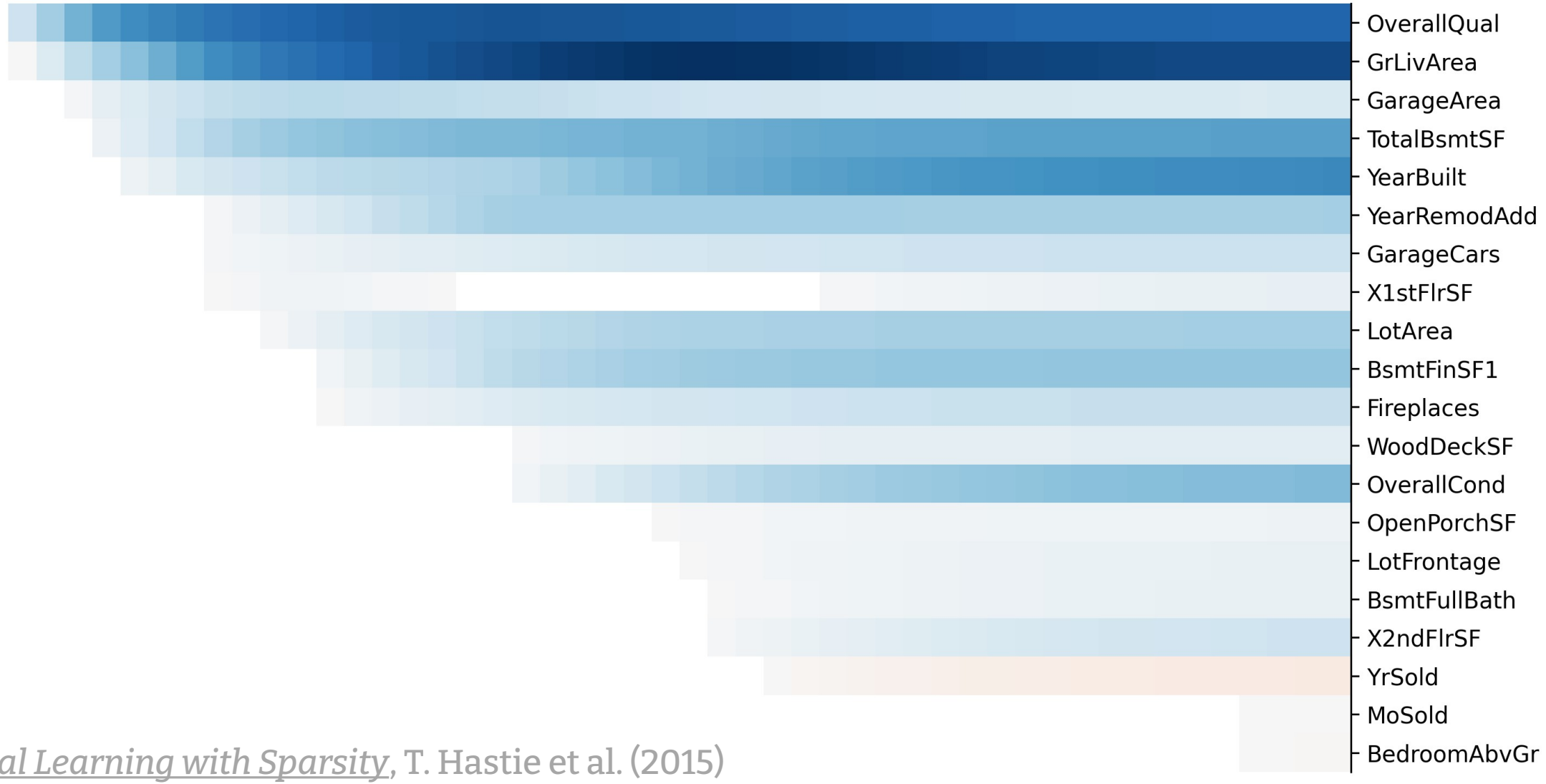
Lasso: regularization path

The output of the lasso is not a single model, but a family of models, increasingly complex: one for each value of λ.



Statistical Learning with Sparsity, T. Hastie et al. (2015)

Lasso: regularization path



Statistical Learning with Sparsity, T. Hastie et al. (2015)

ElasticNet

$$\begin{array}{ll} \text{Find} & w \in \mathbf{R}^k \\ \text{To minimize} & \|Xw - y\|_2^2 + \lambda \|w\|_1 + \mu \|w\|_2^2 \end{array}$$

```
# ElasticNet (linear regression with L¹ and L² penalties)
model = sklearn.linear_model.ElasticNet( alpha = alpha )
model.fit( X, y )
yhat = model.predict(X)
```

Statistical Learning with Sparsity, T. Hastie et al. (2015)

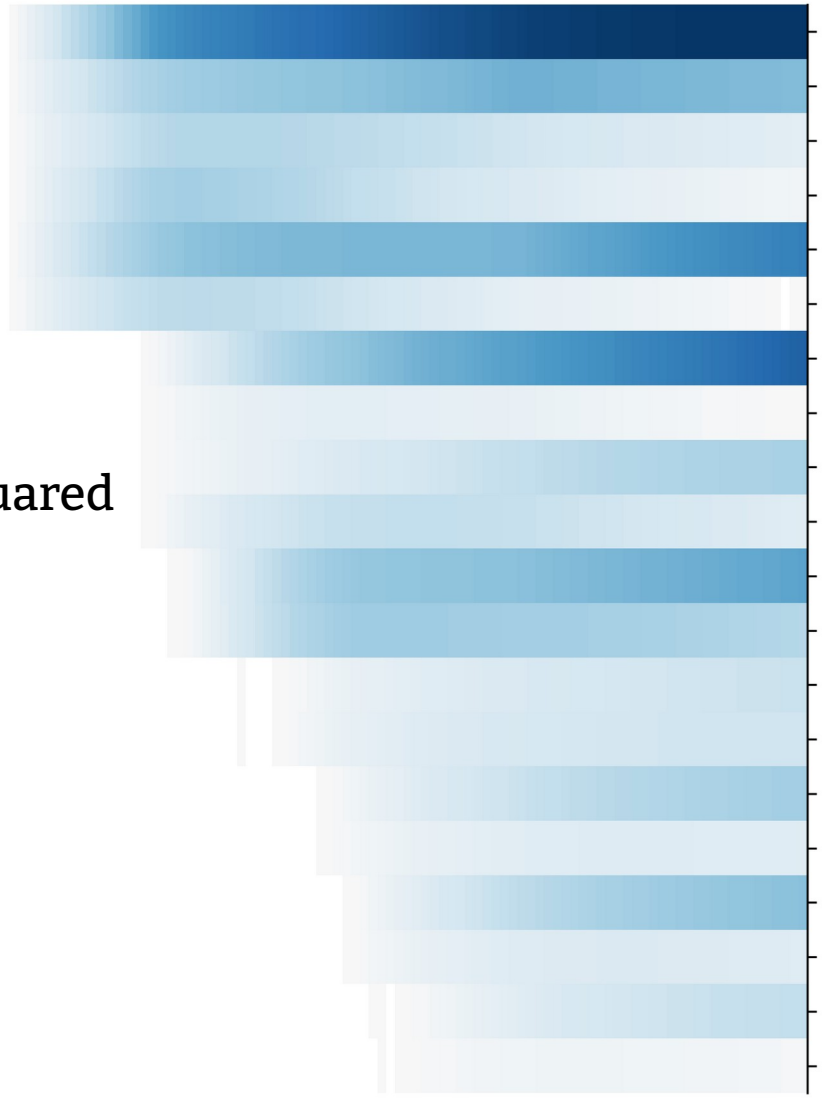
Group lasso

Boolean mask

$$\text{Find} \quad w \in \mathbf{R}^k$$

$$\text{To minimize} \quad \|Xw - y\|_2^2 + \lambda \sum_k \|w \odot m_k\|_2$$

L^2 norm, not squared



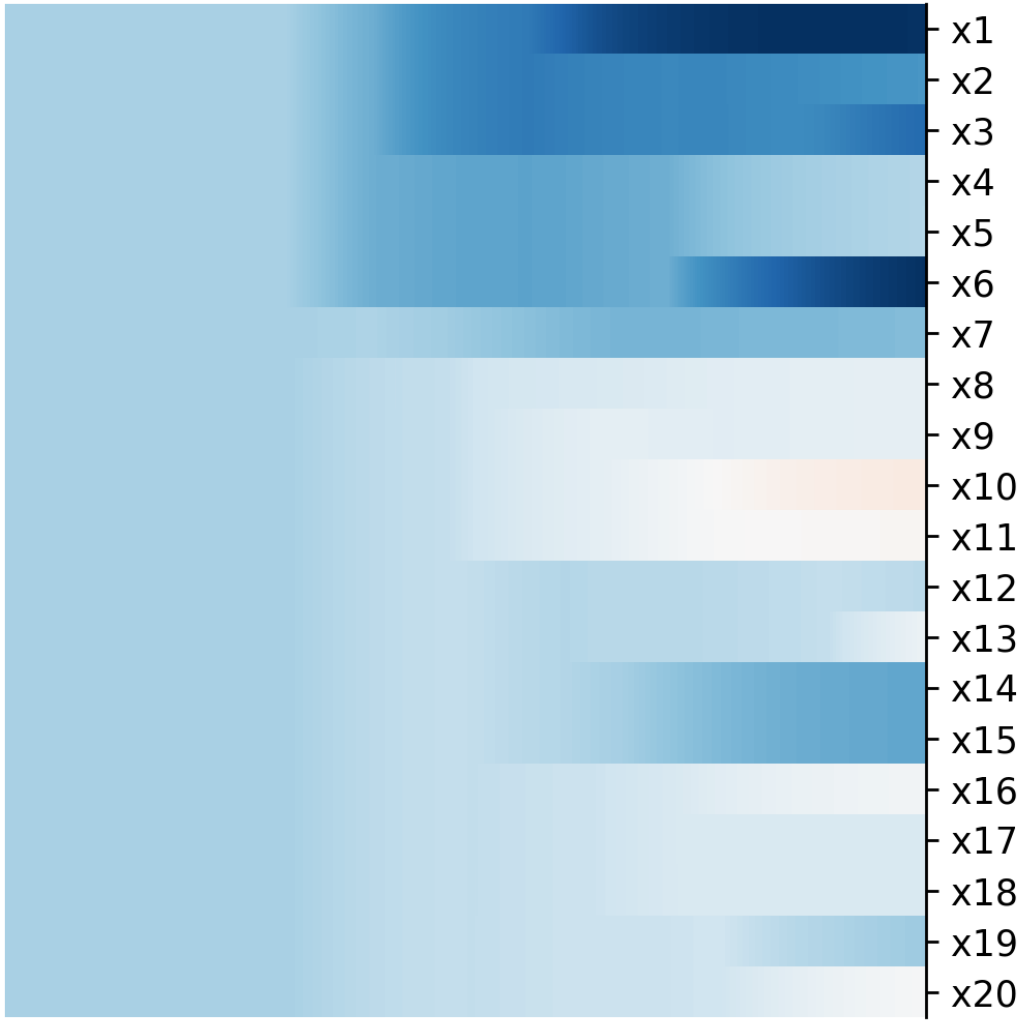
```
# Group lasso (often used for the rows of a matrix)
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - ( intercept + X.values @ w.T )
objective = cp.sum_squares( residuals ) / n
for i in np.unique(c):
    objective += alpha * cp.norm2( cp.multiply( (c==i), w ) )
prob = cp.Problem( cp.Minimize(objective), [] )
result = prob.solve()
w = pd.Series( w.value, index = X.columns )
```

Statistical Learning with Sparsity, T. Hastie et al. (2015)

Fused lasso

$$\begin{aligned} &\text{Find} && w \in \mathbf{R}^k \\ &\text{To minimize} && \|Xw - y\|_2^2 + \lambda \sum_i |w_{i+1} - w_i| \end{aligned}$$

```
# Fused lasso
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - ( intercept + X.values @ w.T )
objective = cp.sum_squares( residuals ) / n + alpha
for i in range(1,k):
    objective += alpha * cp.abs( w[i] - w[i-1] )
prob = cp.Problem( cp.Minimize(objective), [] )
result = prob.solve()
w = pd.Series( w.value, index = X.columns )
```

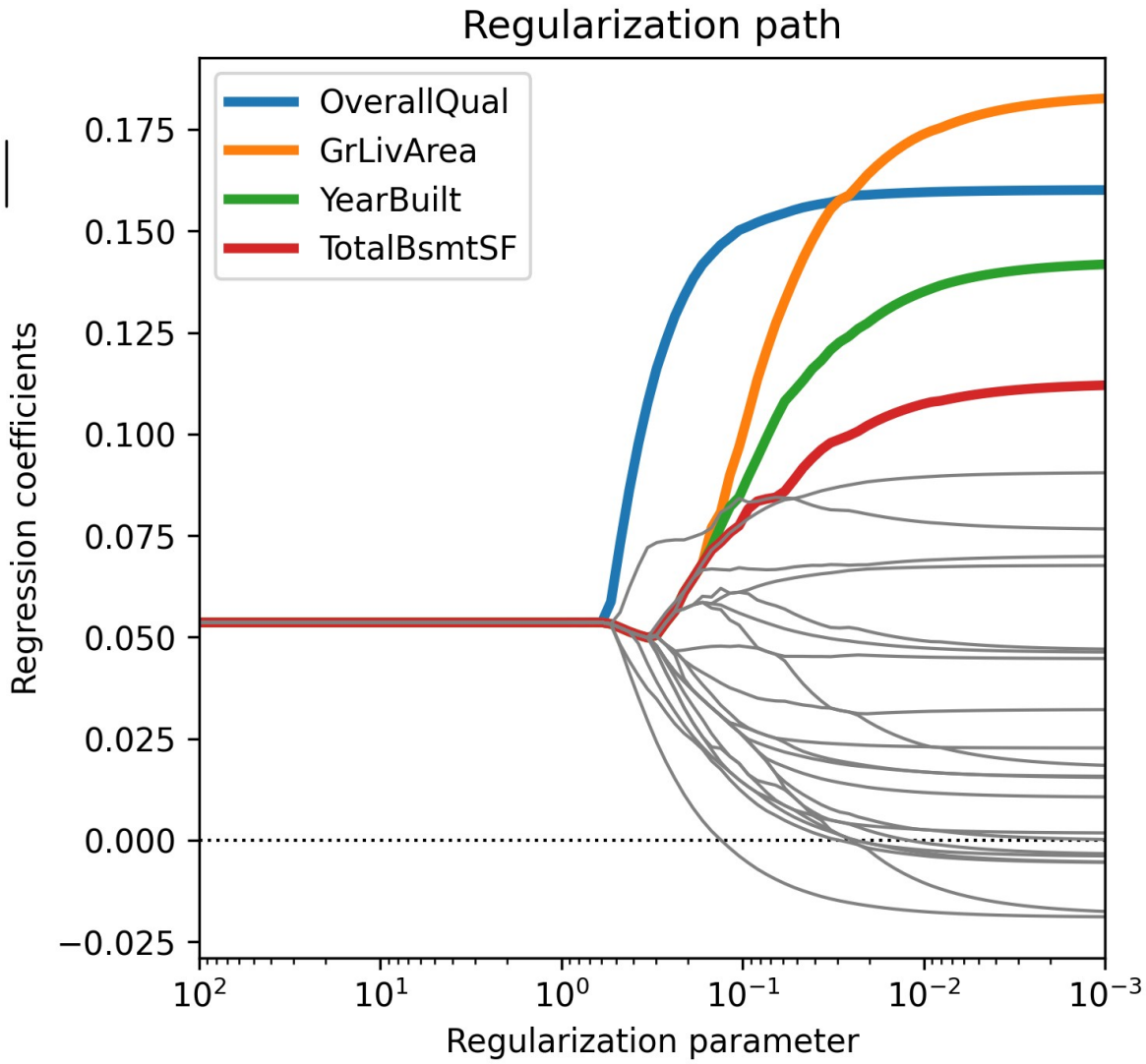


Statistical Learning with Sparsity, T. Hastie et al. (2015)

Graph sparsity

$$\begin{aligned} &\text{Find} && w \in \mathbf{R}^k \\ &\text{To minimize} && \|Xw - y\|_2^2 + \lambda \sum_{(i,j) \in E} |w_i - w_j| \end{aligned}$$

```
# Graph sparsity (complete graph)
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - (intercept + X.values @ w.T)
objective = cp.sum_squares( residuals ) / n
for i,j in g.edges:
    p = alpha * cp.abs( w[i] - w[j] )
    objective += p / len(g.edges)
prob = cp.Problem( cp.Minimize(objective), [] )
result = prob.solve()
w = pd.Series( w.value, index = X.columns )
```



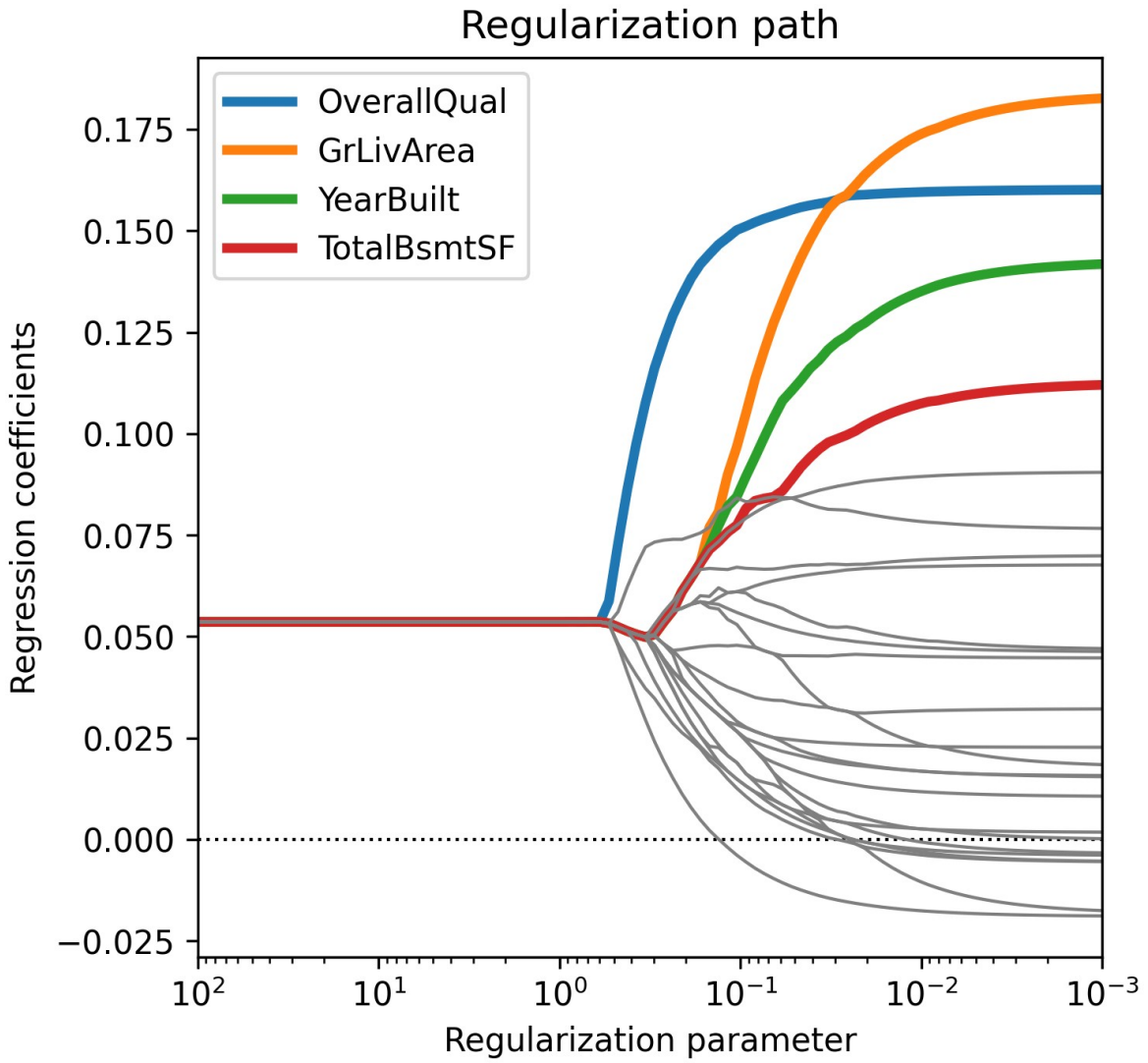
Statistical Learning with Sparsity, T. Hastie et al. (2015)

Graph sparsity

$$\begin{aligned} &\text{Find} && w \in \mathbf{R}^k \\ &\text{To minimize} && \|Xw - y\|_2^2 + \lambda \sum_{i<j} |w_i - w_j| \end{aligned}$$

```
# Graph sparsity (complete graph)
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - (intercept + X.values @ w.T)
objective = cp.sum_squares( residuals ) / n
for i in range(k):
    for j in range(i+1,k):
        p = alpha * cp.abs( w[i] - w[j] )
        objective += p / k / (k-1) * 2
prob = cp.Problem( cp.Minimize(objective), [] )
result = prob.solve()
w = pd.Series( w.value, index = X.columns )
```

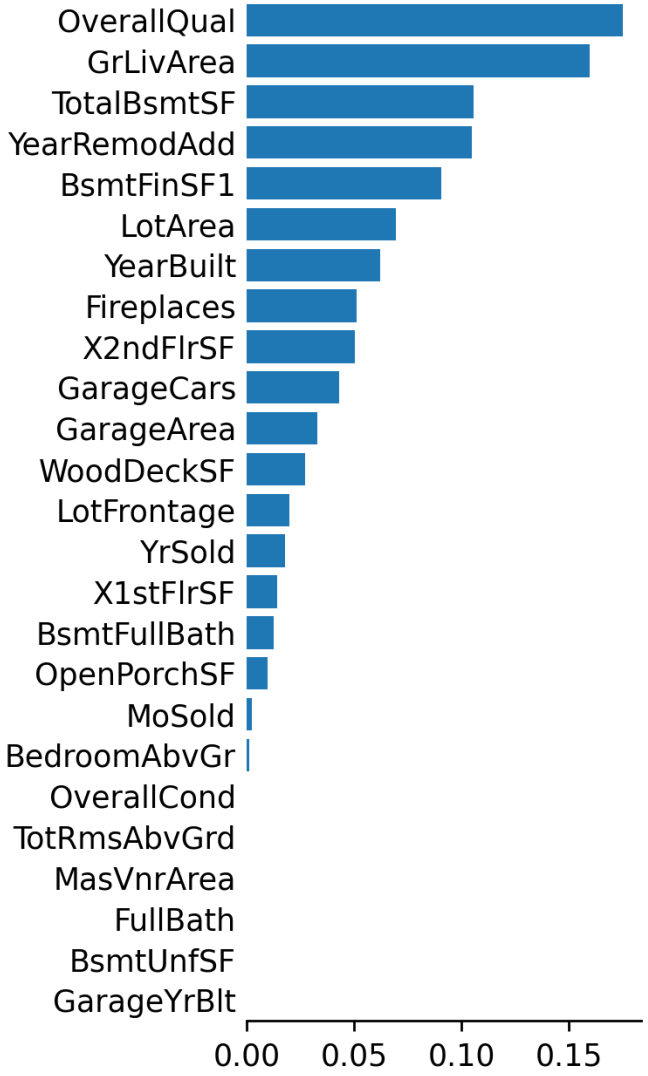
Statistical Learning with Sparsity, T. Hastie et al. (2015)



Sign constraints

$$\begin{aligned} \text{Find} \quad & w \in \mathbf{R}^k \\ \text{To minimize} \quad & \|Xw - y\|_2^2 \\ \text{Such that} \quad & \forall i \; w_i \geqslant 0 \end{aligned}$$

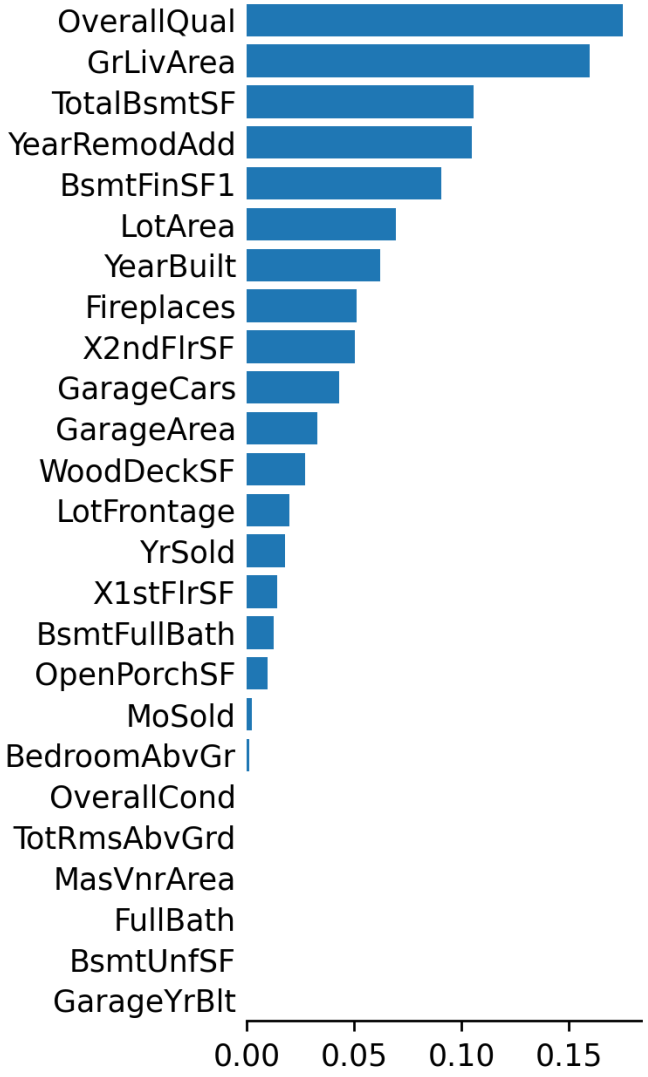
```
# Linear regression with sign constraints
model = LinearRegression( positive = True )
model.fit(X, y)
yhat = model.predict(X)
```



Sign constraints

$$\begin{aligned} \text{Find} \quad & w \in \mathbf{R}^k \\ \text{To minimize} \quad & \|Xw - y\|_2^2 \\ \text{Such that} \quad & \forall i \; w_i \geqslant 0 \end{aligned}$$

```
# Linear regression with sign constraints
n, k = X.shape
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - (intercept + (X * signs).values @ w.T)
objective = cp.sum_squares( residuals ) / n
constraints = [ w >= 0 ]
prob = cp.Problem( cp.Minimize(objective), constraints )
result = prob.solve()
w = pd.Series( w.value, index = X.columns )
```



SLIM

- Combine binary inputs with simple integer coefficients

$$
\begin{aligned}
&6 \times 1_{\text{peritonitis=generalized}} + \\
&6 \times 1_{\text{appendix diameter=9-18 mm}} + \\
&5 \times 1_{\text{appendix diameter=6-9 mm}} + \\
&4 \times 1_{\text{appendix on ultrasound}} + \\
&2 \times 1_{\text{peritonitis=local}}
\end{aligned}
$$

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

SLIM

$$\text{Find} \quad \beta \in [\![-10, 10,]\!]^k$$

$$\text{To minimize} \quad \sum_i \text{loss}\left(y_i, \text{sign}(\beta' x_i)\right) + \lambda \|\beta\|_0 + \epsilon \|\beta\|_1$$

We want many of the β_i to be 0

0-1 loss

Binary output: ±1

Binary inputs: ±1

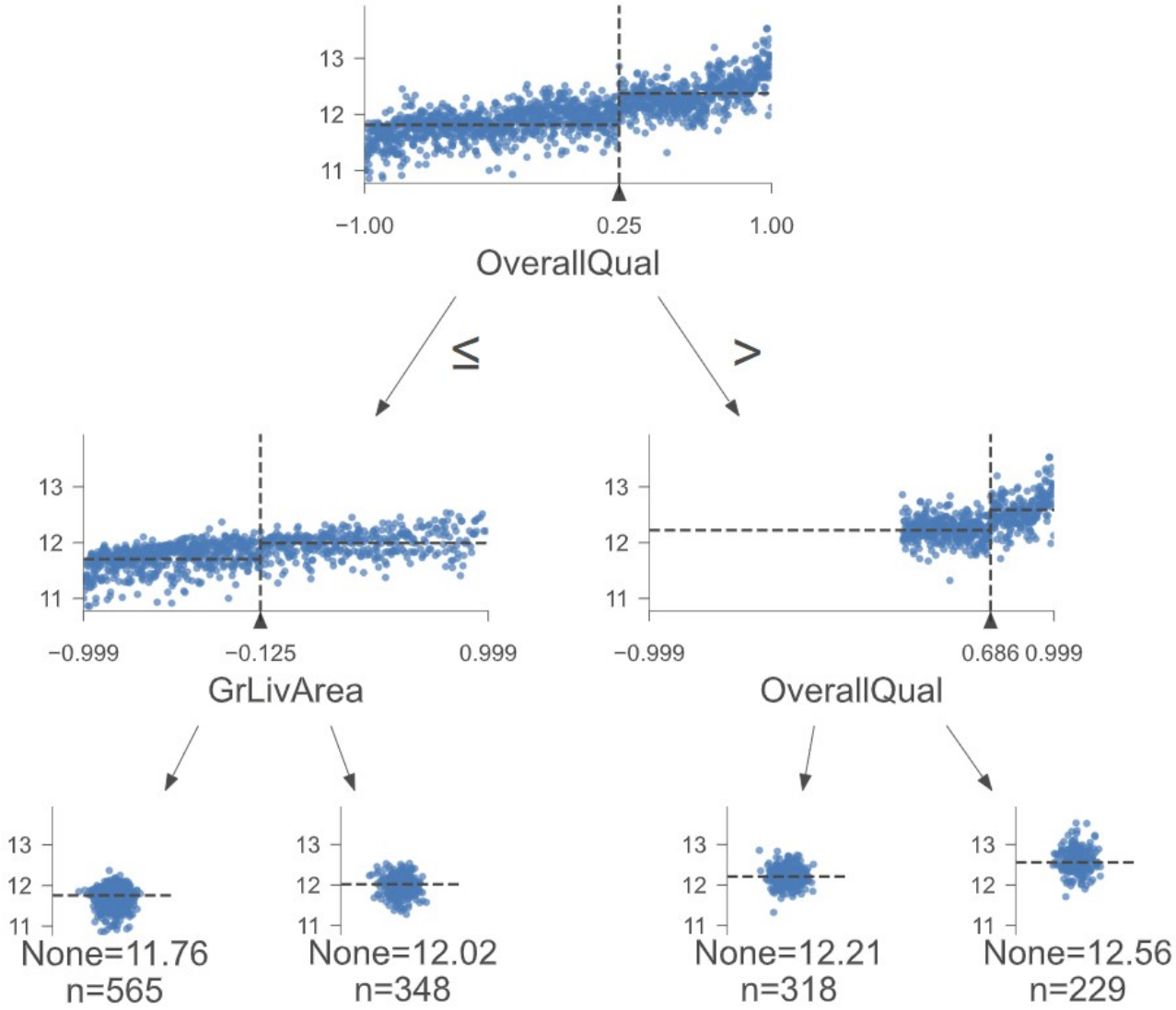
Small ℓ^1 penalty

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

Bayesian linear regression

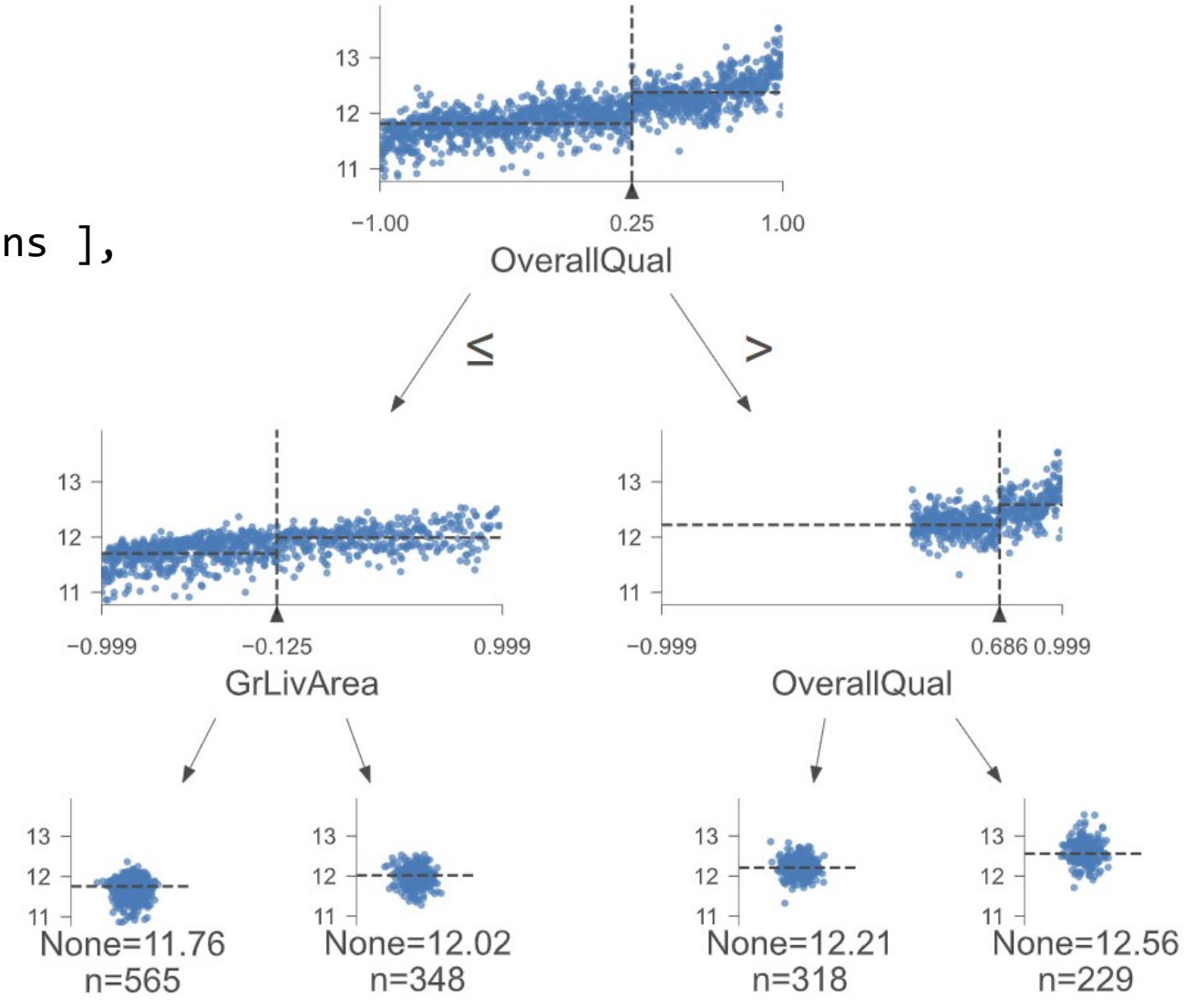
Machine Learning

Decision tree

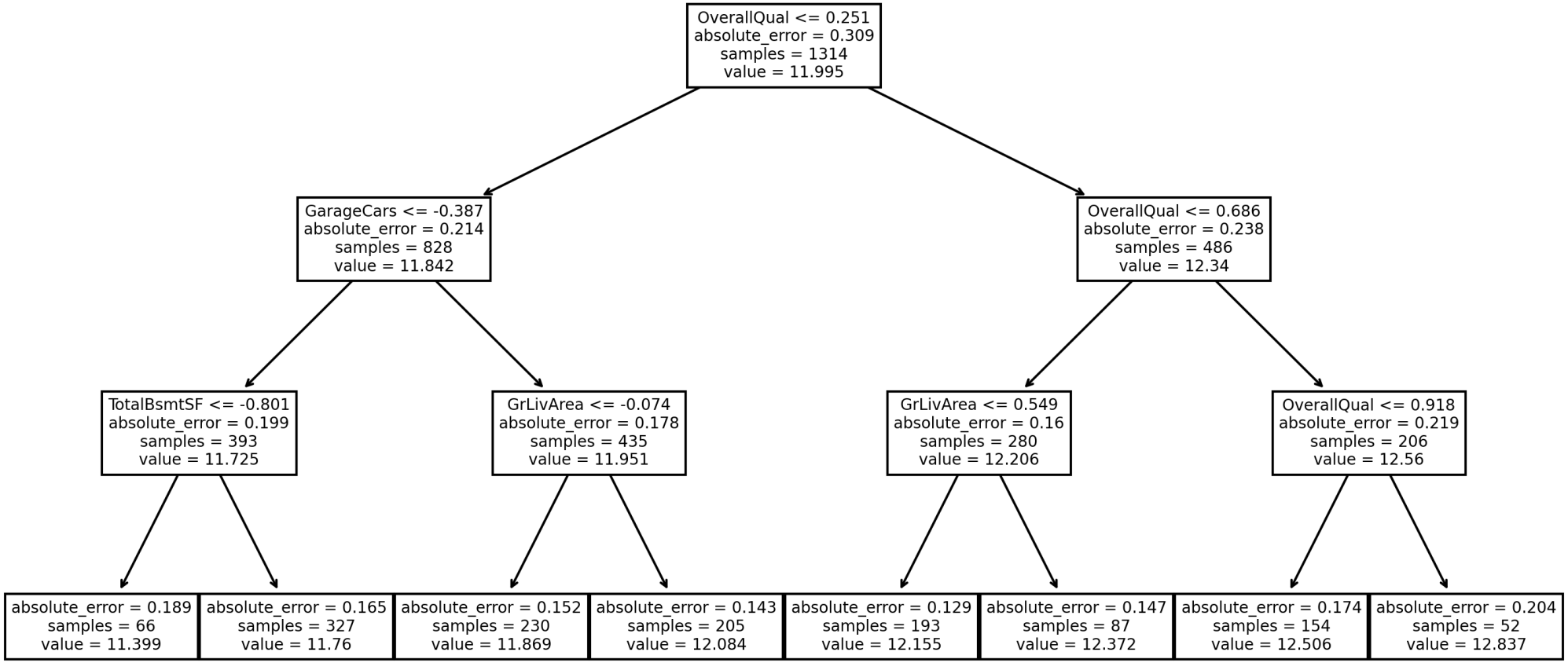


Monotonic decision tree

```
# Decision tree with a monotonicity constraint
model = sklearn.tree.DecisionTreeRegressor(
    monotonic_cst = [ signs[u] for u in X.columns ],
)
model.fit( X, y )
```

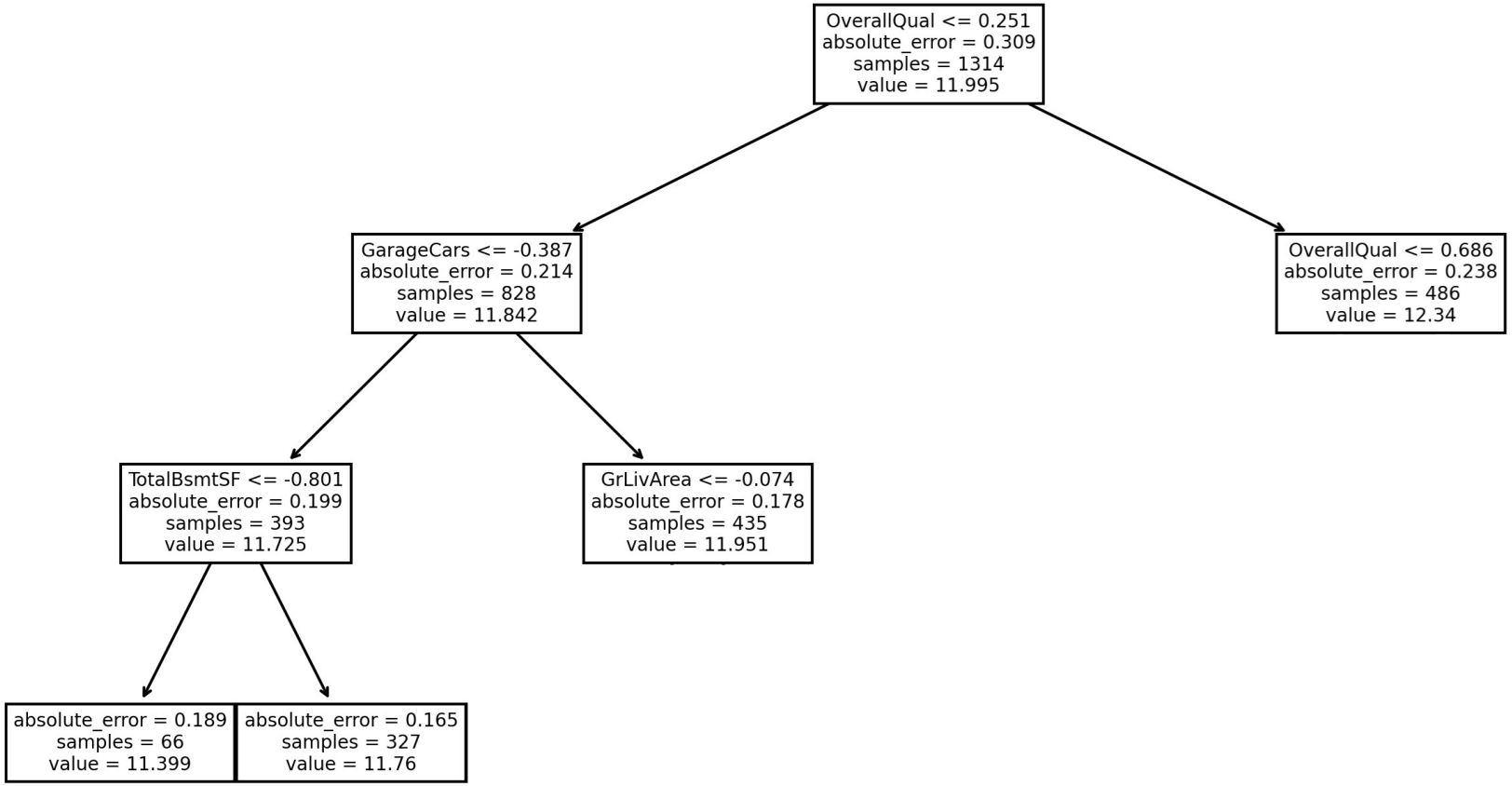


Falling rule lists



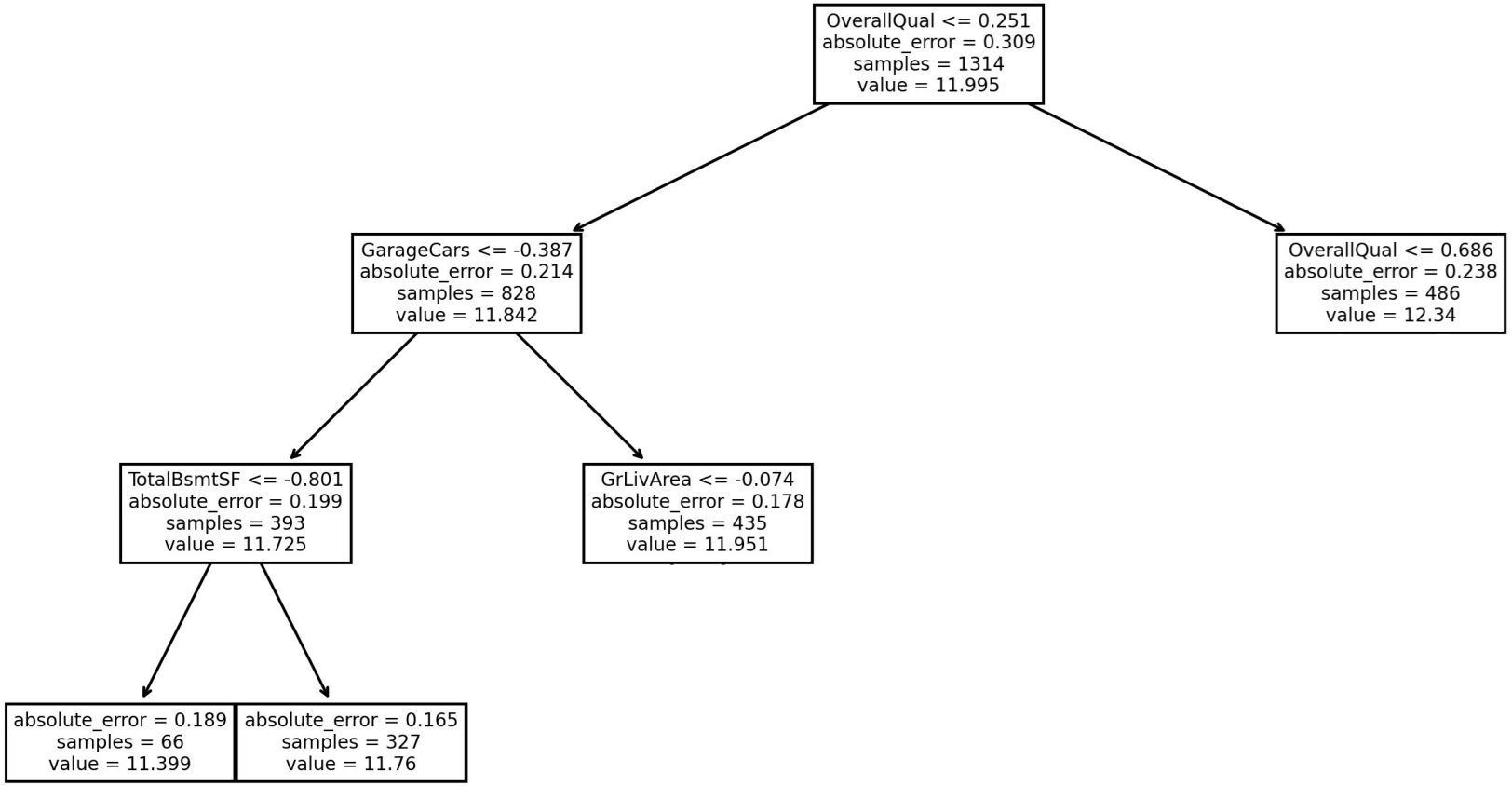
An optimization approach to learning falling rule lists, C. Chen and C. Rudin (2017)

Falling rule lists



An optimization approach to learning falling rule lists, C. Chen and C. Rudin (2017)

Falling rule lists

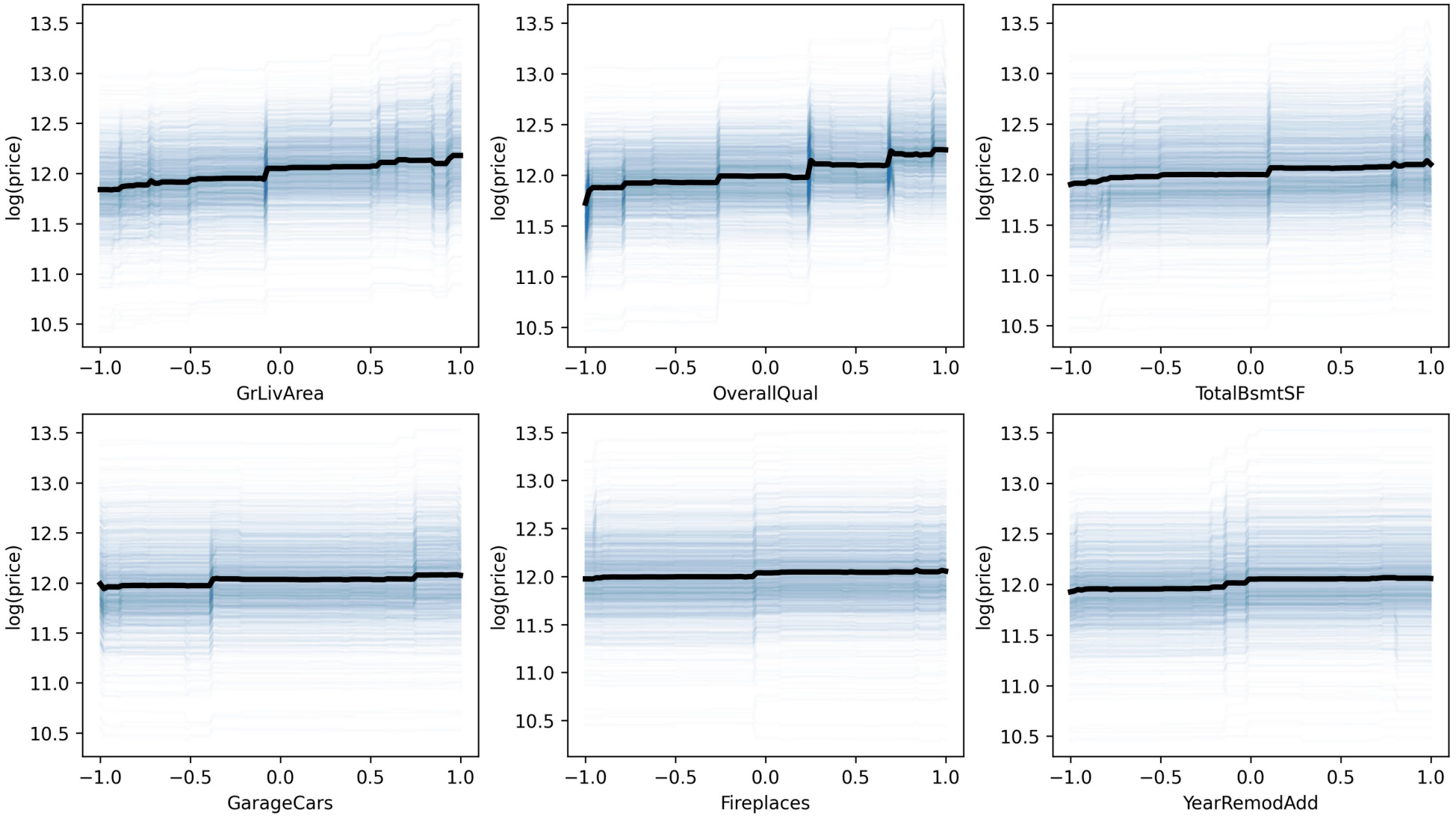


An optimization approach to learning falling rule lists, C. Chen and C. Rudin (2017)

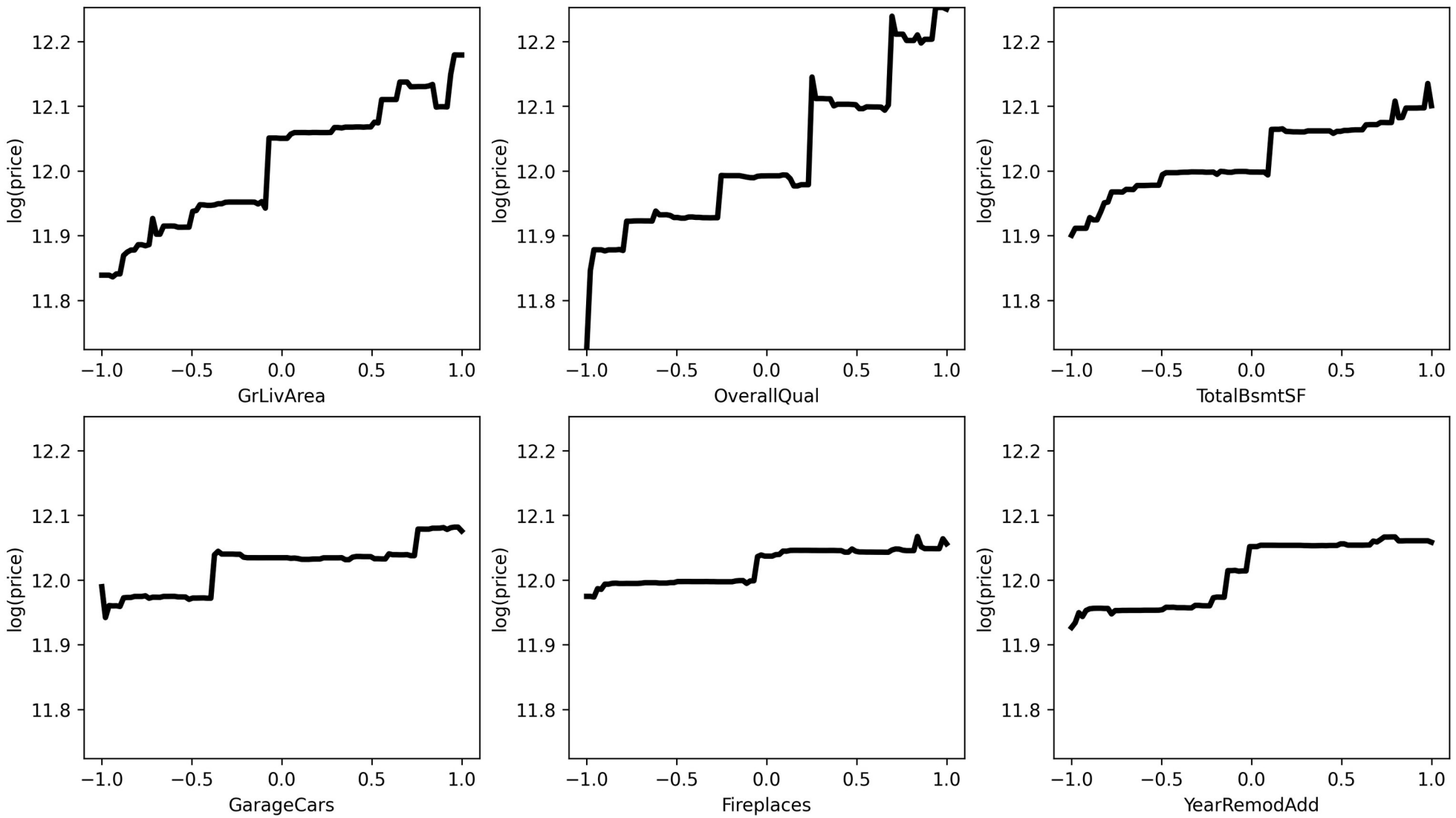
Boosting

- Fit a simple (“base”) model to the data
- Fit another base model to misclassified data
- Combine them
- Iterate
- The final model is a linear combination of base models

Gradient boosting



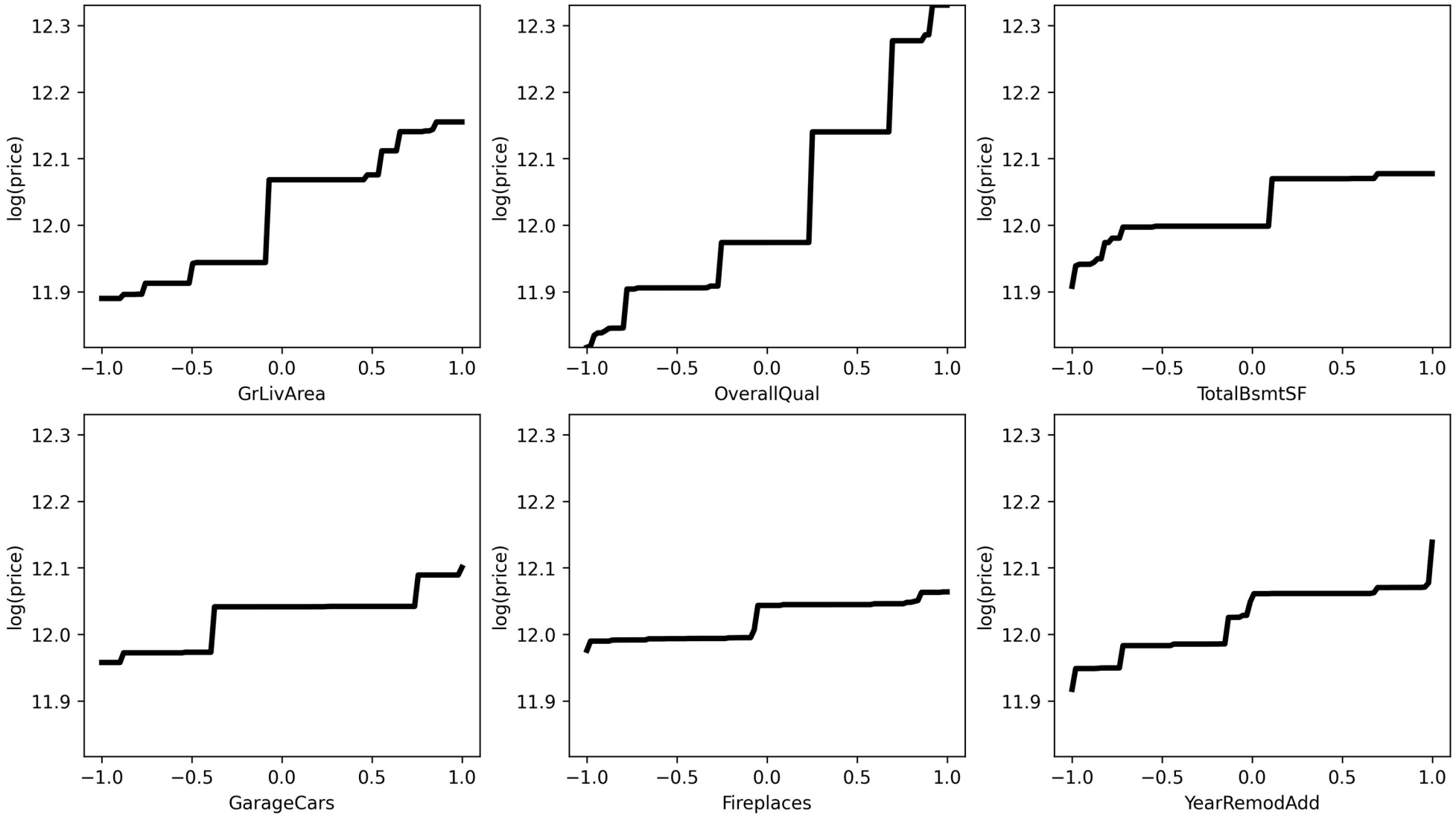
Gradient boosting



Gradient boosting

```
# Gradient boosting with a monotonicity constraint

model = xgboost.XGBRegressor(
    monotone_constraints = { k: int(v) for k,v in signs.items() },
)

model = lightgbm.LGBMRegressor(
    monotone_constraints = [ signs[u] for u in X.columns ],
)

model = catboost.CatBoostRegressor(
    verbose = False,
    monotone_constraints = [ signs[u] for u in X.columns ],
)
```

Gradient boosting



GAM

Find functions $f_1,\ldots,f_k$ such that

$$y \approx f_1(x_1) + \cdots + f_k(x_k)$$

The f_i's are often modeled with splines, and "backfitted":

$$f_i(x_i) \approx y - \sum_{j \neq i} f_j(x_j)$$

Generalized additive models (S.N. Wood, 2017)

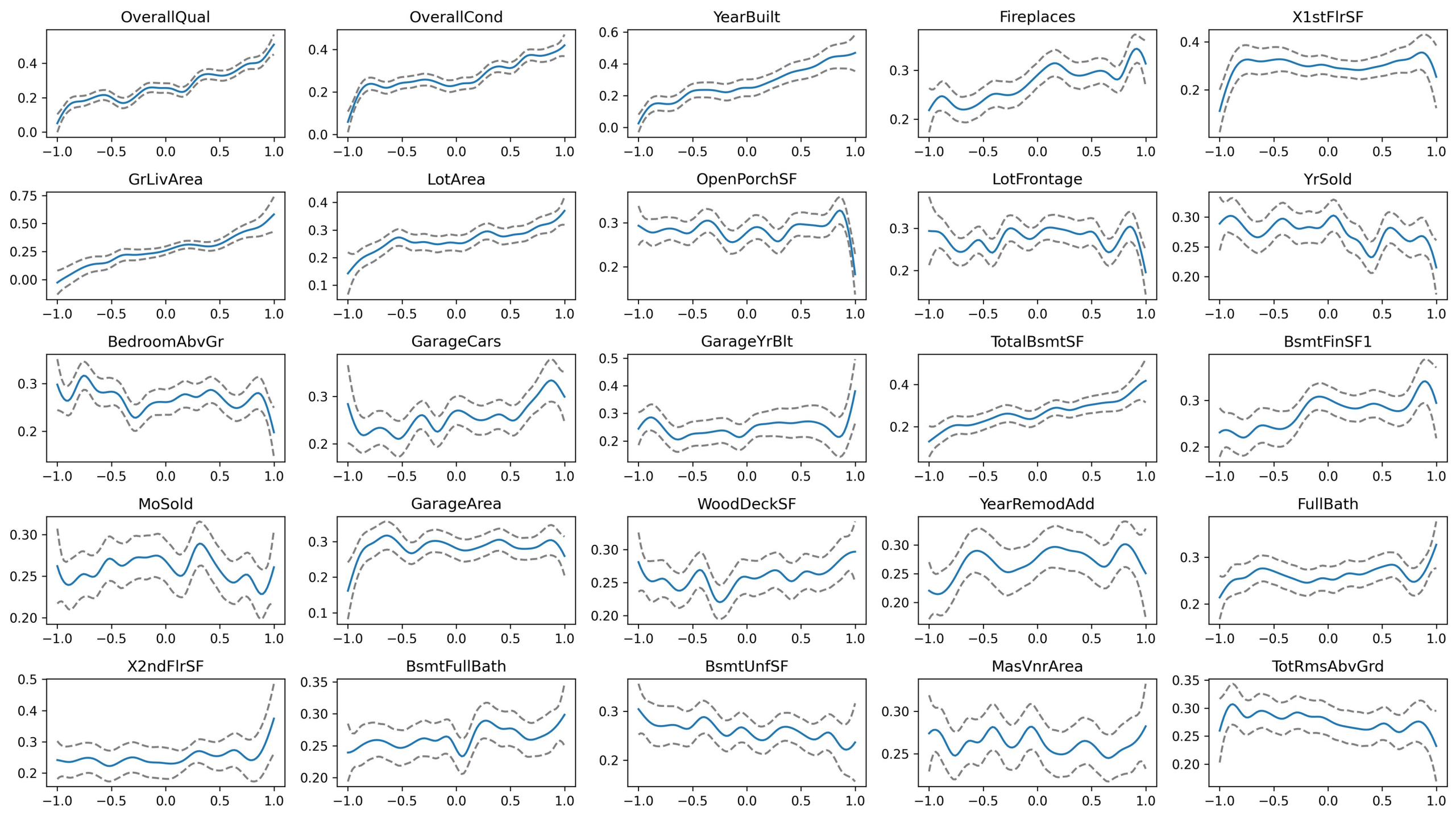
GAM

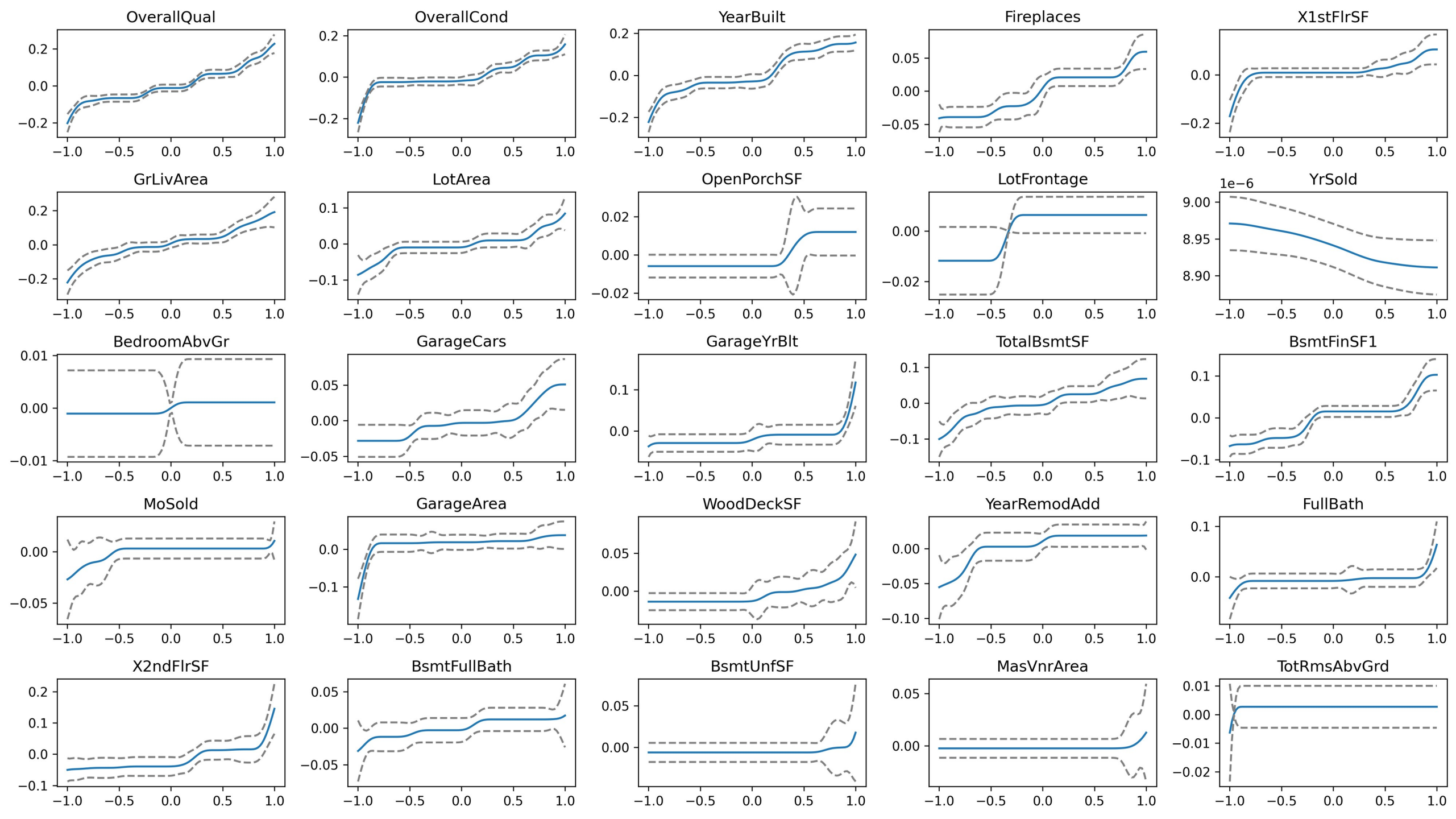
Find functions $f_1,\ldots,f_k$ such that

$$y \approx f_1(x_1) + \cdots + f_k(x_k)$$

```
# Generalized additive model (GAM) with a monotonicity constraint
constraints = [
    'monotonic_inc' if signs[u] > 0 else 'monotonic_dec'
    for u in X.columns
]
gam = LinearGAM(constraints=constraints).fit(X,y)
```

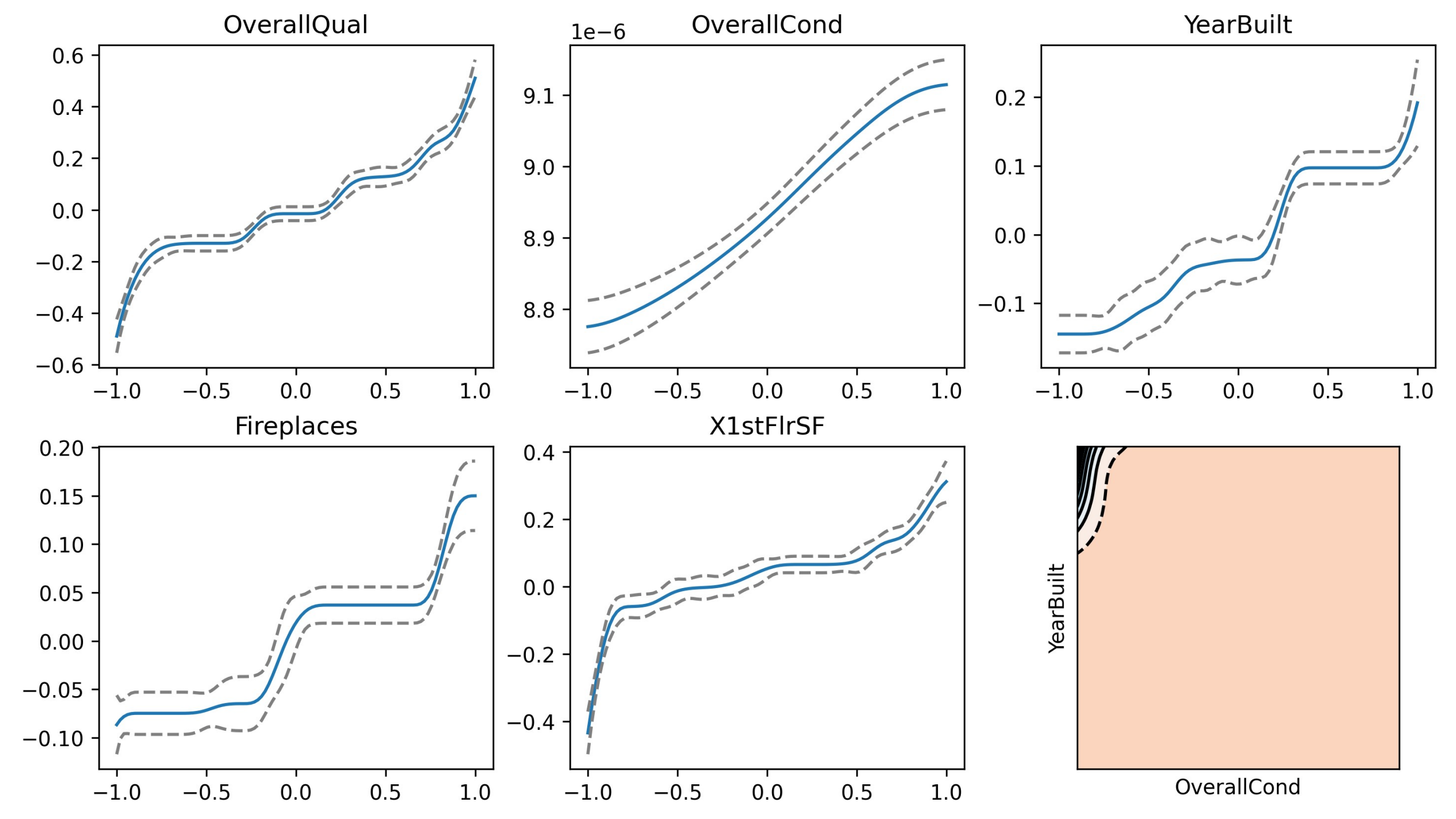
https://pygam.readthedocs.io/en/latest/


OverallQual
OverallCond
YearBuilt
Fireplaces
X1stFlrSF
GrLivArea
LotArea
OpenPorchSF
LotFrontage
YrSold
BedroomAbvGr
GarageCars
GarageYrBlt
TotalBsmtSF
BsmtFinSF1
MoSold
GarageArea
WoodDeckSF
YearRemodAdd
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF
MasVnrArea
TotRmsAbvGrd


OverallQual
OverallCond
YearBuilt
Fireplaces
X1stFlrSF
GrLivArea
LotArea
OpenPorchSF
LotFrontage
1e−6
YrSold
BedroomAbvGr
GarageCars
GarageYrBlt
TotalBsmtSF
BsmtFinSF1
MoSold
GarageArea
WoodDeckSF
YearRemodAdd
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF
MasVnrArea
TotRmsAbvGrd

GAM with interactions (GA²M)

$$y \approx \sum_i f_i(x_i) + \sum_{i<j} g_{ij}(x_i, x_j)$$


OverallQual
OverallCond
1e−6
YearBuilt
Fireplaces
X1stFlrSF
YearBuilt
OverallCond

GAM Boosting

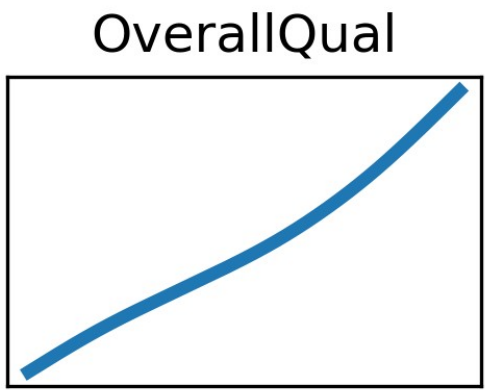
Boosting with GAMs as base learners:

- The variables enter the model one by one
- They are transformed in increasingly complex (nonlinear) ways

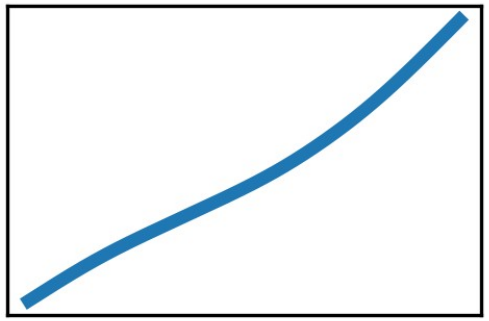
This gives a GAM *regularization path*.

```
# GAM Boosting (this is R code)
mboost::gamboost( y ~ ., data = d )
```

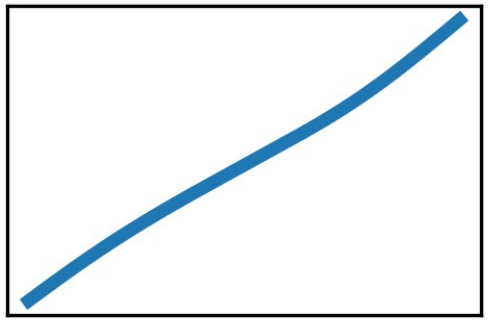
Model-based boosting in R: a hands-on tutorial using the R package mboost (B. Hofner et al., 2014)

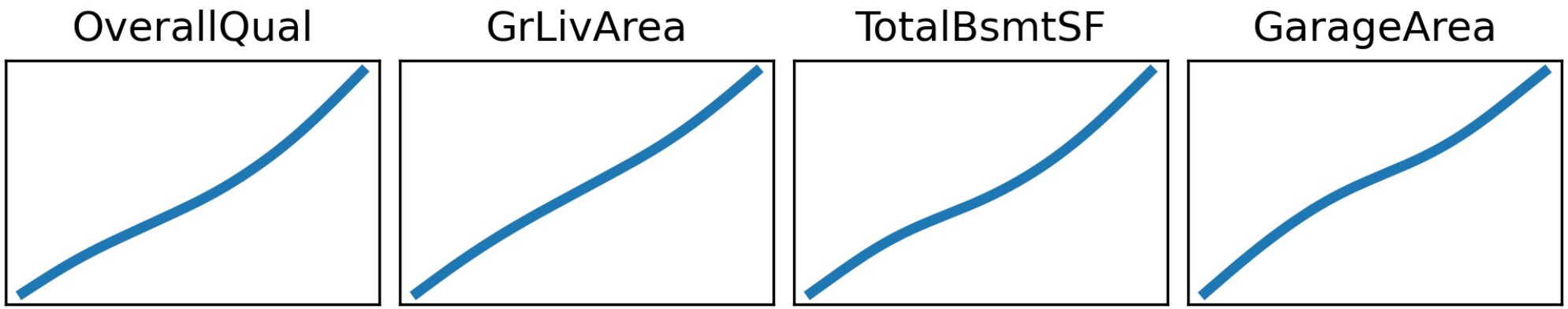

OverallQual

OverallQual

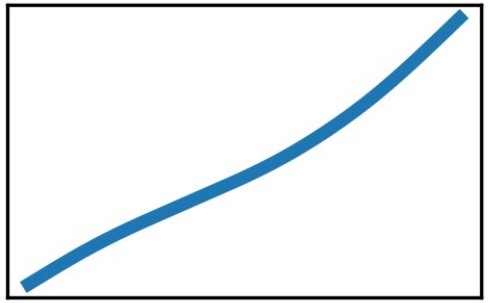


GrLivArea

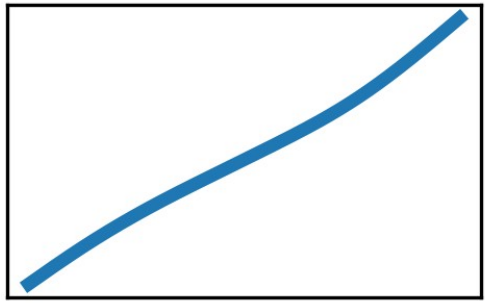



OverallQual
GrLivArea
TotalBsmtSF
GarageArea

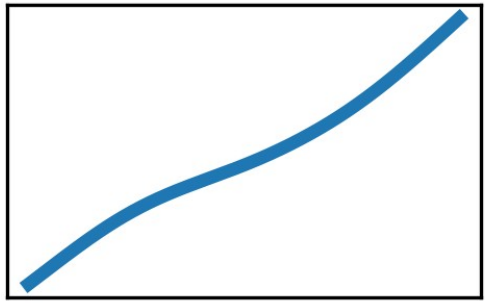
OverallQual



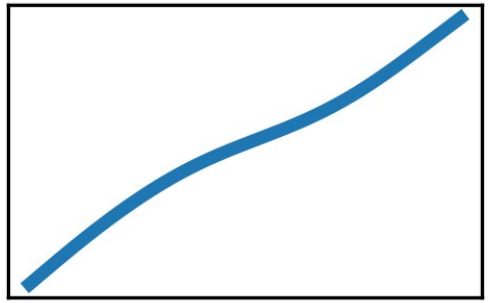
GrLivArea



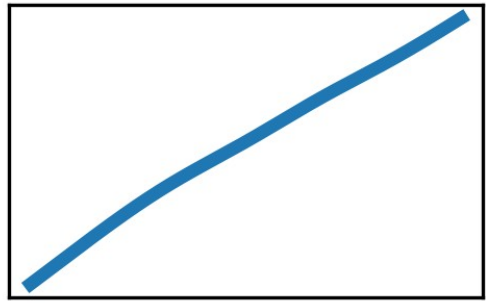
TotalBsmtSF



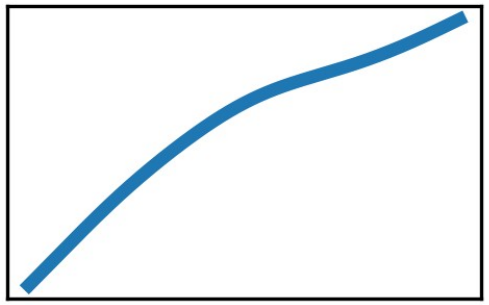
GarageArea



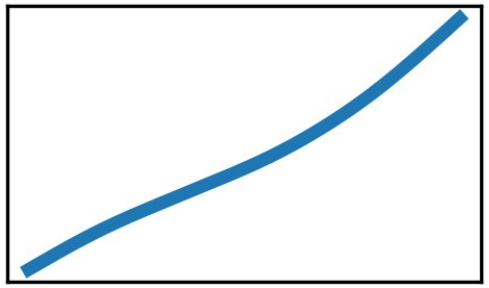
YearBuilt



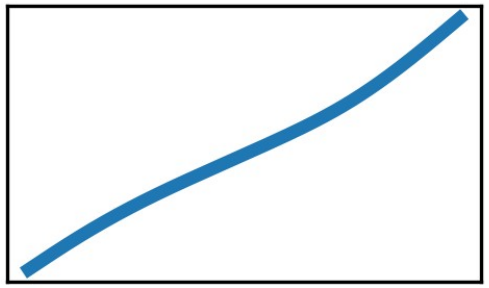
YearRemodAdd



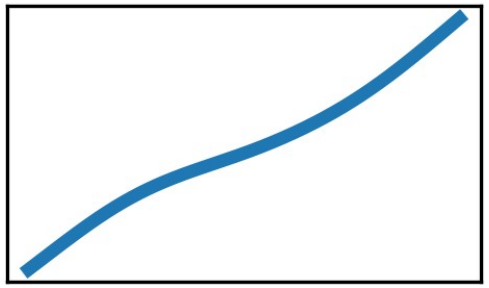
OverallQual



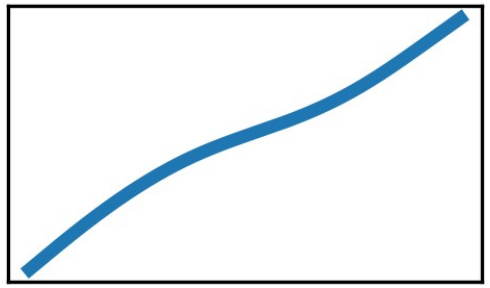
GrLivArea



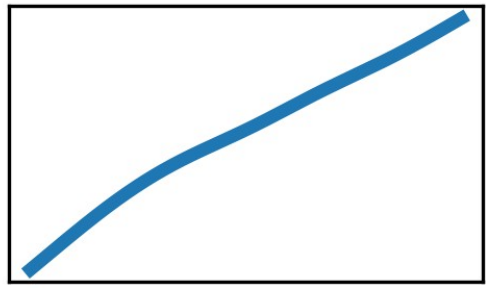
TotalBsmtSF



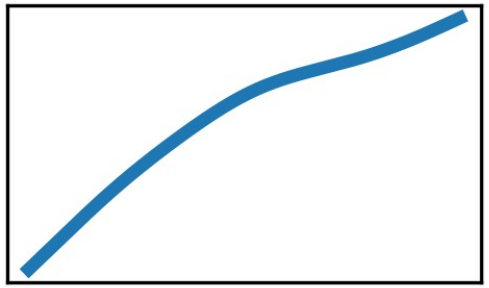
GarageArea



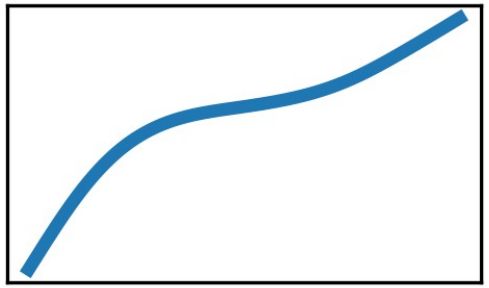
YearBuilt



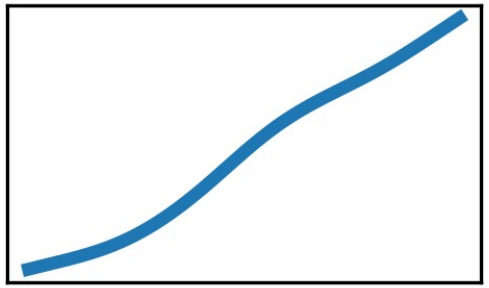
YearRemodAdd



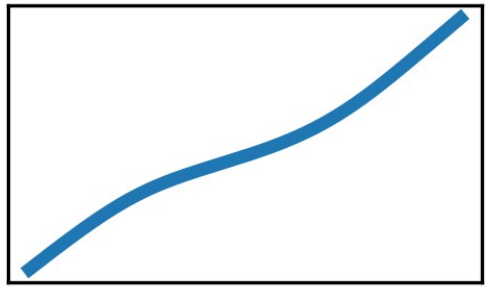
OverallCond



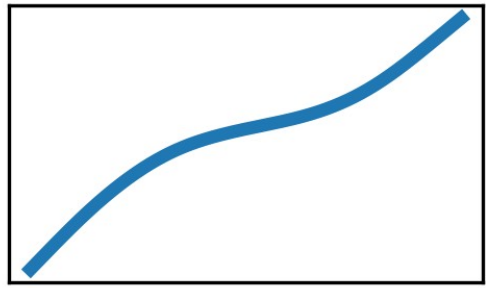
Fireplaces



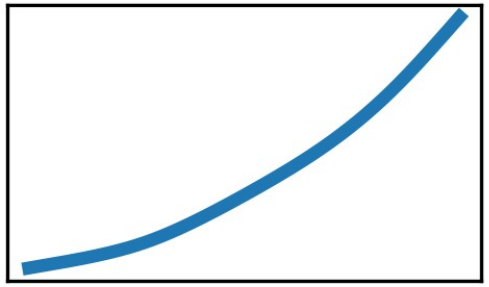
LotArea

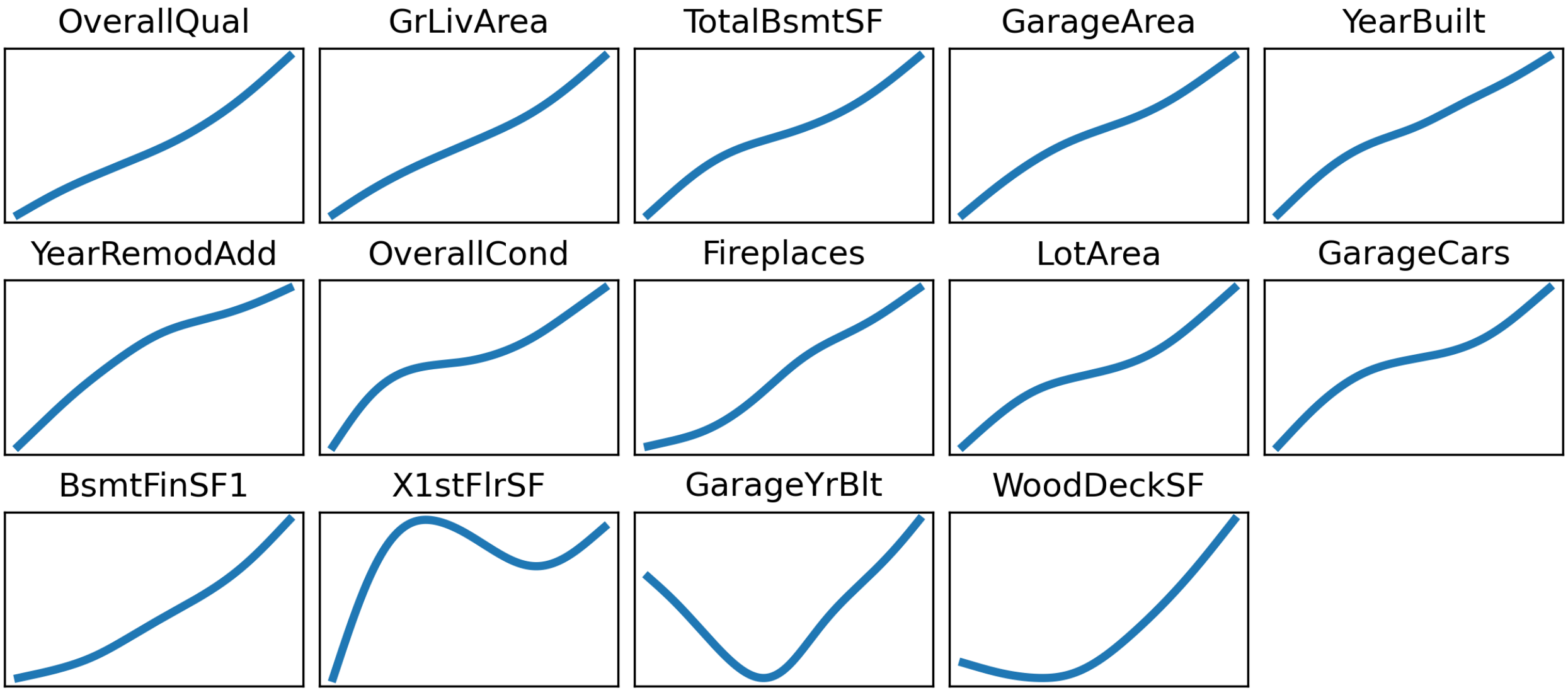


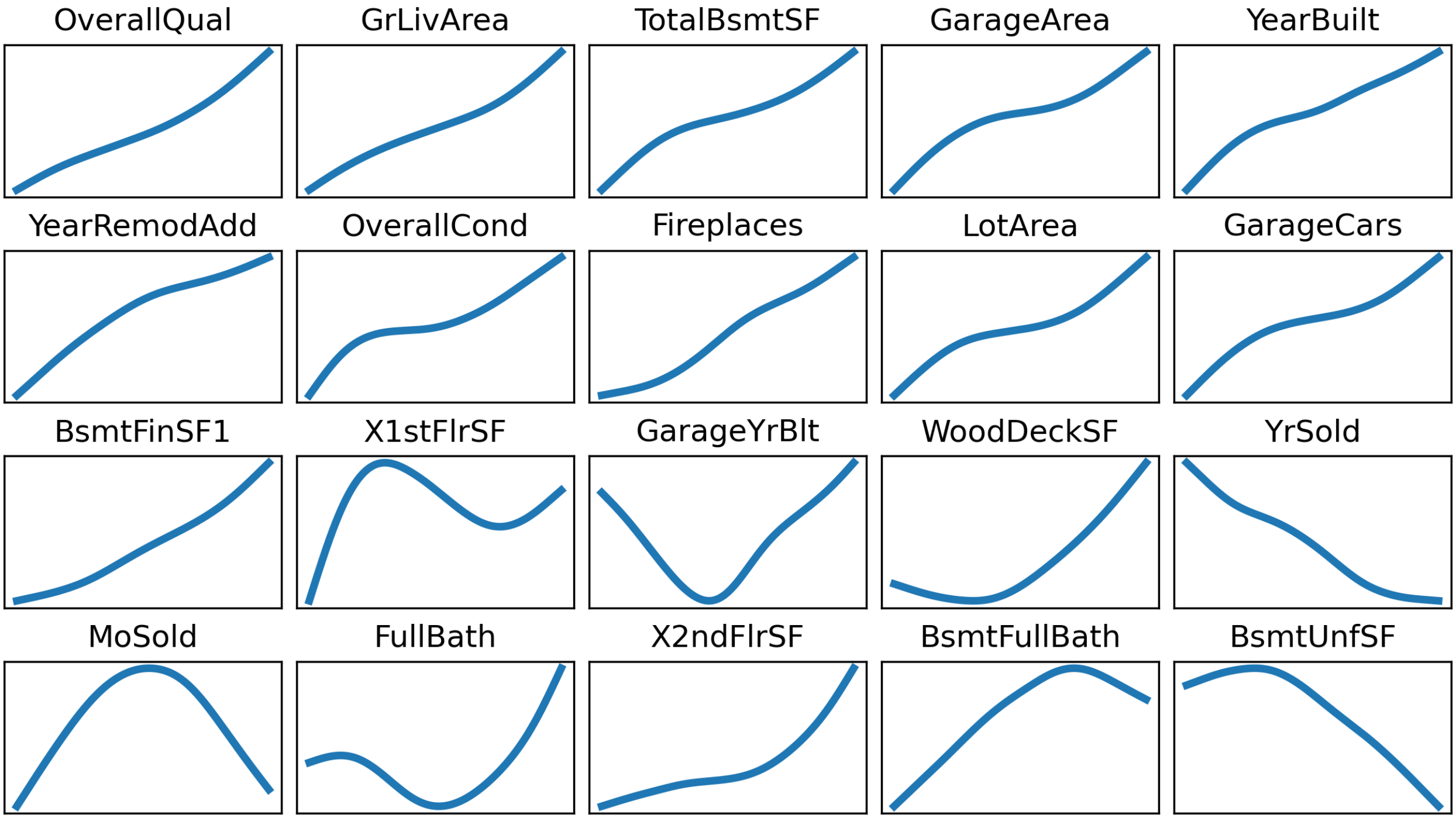
GarageCars

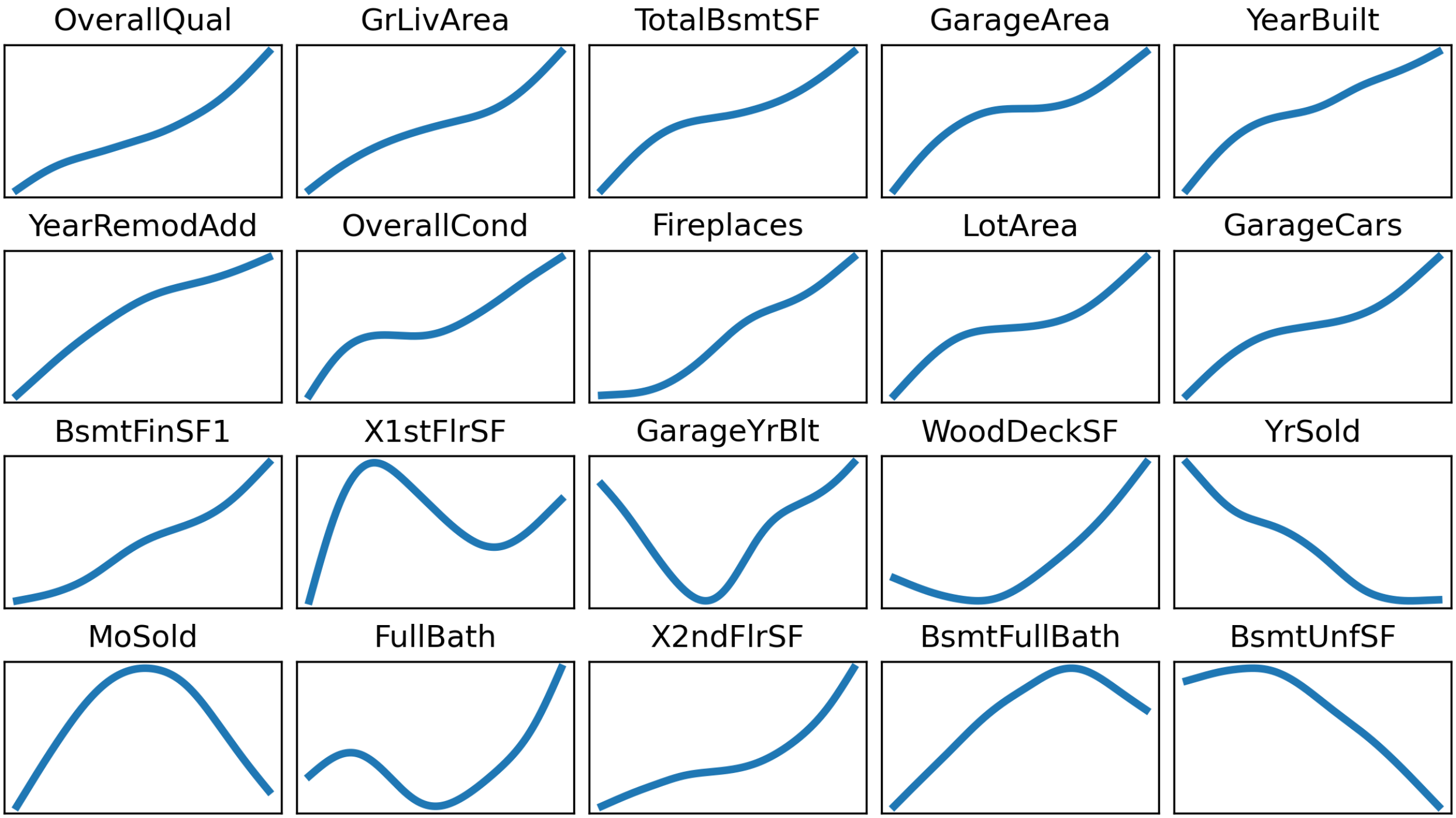


BsmtFinSF1




OverallQual
GrLivArea
TotalBsmtSF
GarageArea
YearBuilt
YearRemodAdd
OverallCond
Fireplaces
LotArea
GarageCars
BsmtFinSF1
X1stFlrSF
GarageYrBlt
WoodDeckSF


OverallQual
GrLivArea
TotalBsmtSF
GarageArea
YearBuilt
YearRemodAdd
OverallCond
Fireplaces
LotArea
GarageCars
BsmtFinSF1
X1stFlrSF
GarageYrBlt
WoodDeckSF
YrSold
MoSold
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF


OverallQual
GrLivArea
TotalBsmtSF
GarageArea
YearBuilt
YearRemodAdd
OverallCond
Fireplaces
LotArea
GarageCars
BsmtFinSF1
X1stFlrSF
GarageYrBlt
WoodDeckSF
YrSold
MoSold
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF

Additive index models

$$\text{GAM} \quad y \approx f_1(x_1) + \cdots + f_k(x_k)$$

$$\text{AIM} \quad y \approx f_1(\beta_1^\top x) + \cdots + f_k(\beta_k^\top x)$$

For better interpretability:

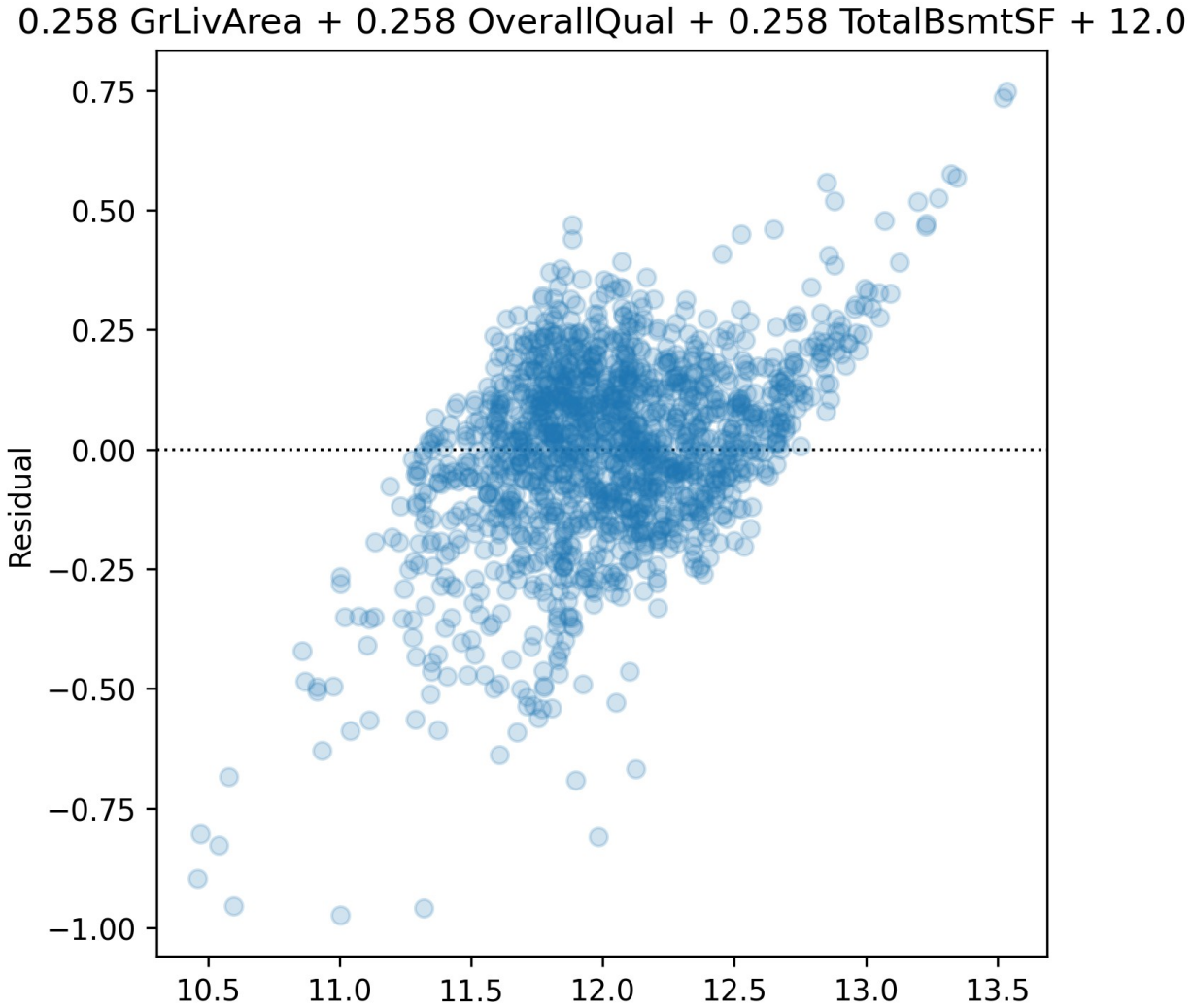
- Add an orthogonality penalty on beta
- Add a sparsity penalty

```python
class AIM(nn.Module):
    """
    Additive Index Model:
      y ≈ f₁(β₁'x) + ⋯ + fₖ(βₖ'x)
    Model f₁,...,fₖ as neural nets (I used one hidden layer and ReLU activations)
    """

    def __init__(self, n_input, n_terms, n_nodes, activation = F.relu):
        super(AIM, self).__init__()
        self.fc = nn.Linear(n_input, n_terms, bias=False)
        self.W1 = nn.Parameter( torch.Tensor(n_terms, n_nodes) )
        self.b1 = nn.Parameter( torch.Tensor(n_terms, n_nodes) )
        self.W2 = nn.Parameter( torch.Tensor(n_terms, n_nodes) )
        self.b = nn.Parameter( torch.Tensor(1) )
        self.activation = activation
        self.reset_parameters()

    def reset_parameters(self):
        for p in self.parameters():
            nn.init.uniform_(p, -1, 1)

    def project(self, x):
        return self.fc(x)  # βᵢ'x

    def transform(self, h):
        h = torch.einsum( 'ik,kl->ikl', h, self.W1 )  # No summation
        h = h + self.b1
        h = self.activation(h)
        h = torch.einsum( 'ikl,kl->ik', h, self.W2 )  # fᵢ(βᵢ'x)
        return h

    def forward(self, x):
        h = self.project(x)
        h = self.transform(h)
        h = torch.einsum( 'ik->i', h )
        h = h + self.b
        return h
```

Symbolic regression

Find a simple formula fitting the data, using +, -, ×, /, ^, log, exp, sin, cos.

Works best in physics, with few variables and lots of data.



Interpretable machine learning for science with PySR and SymbolicRegression.jl, M. Cranmer (2023)

Code: symbolic regression

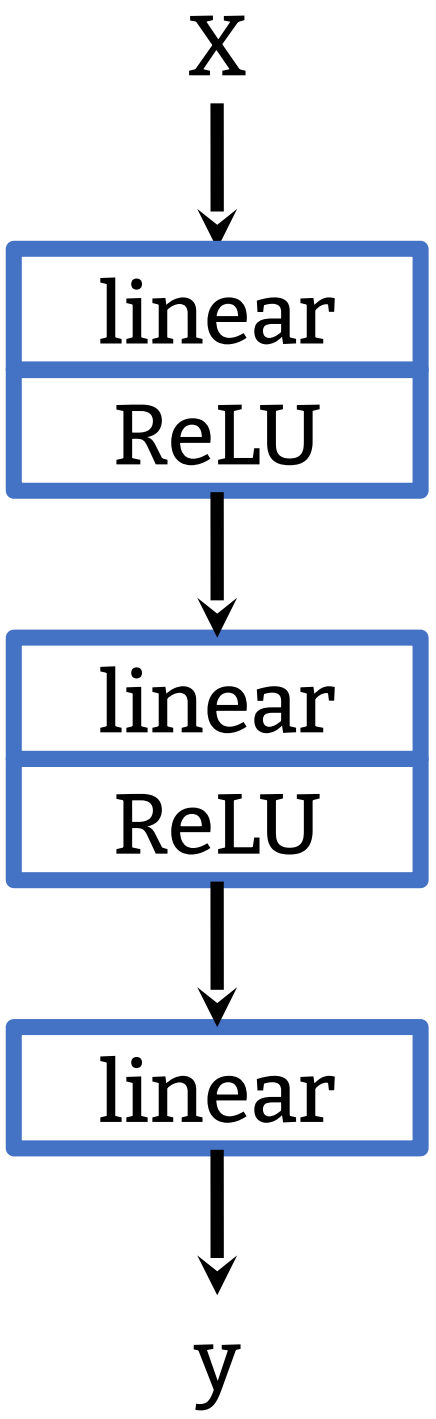
```
# Symbolic regression (example from the documentation)
model = PySRRegressor(
    niterations = 40,
    binary_operators = ["+", "*"],
    unary_operators = [
        "cos",
        "exp",
        "sin",
        "inv(x) = 1/x",
    ],
    extra_sympy_mappings = {"inv": lambda x: 1 / x},
    elementwise_loss = "loss(prediction, target) = (prediction - target)^2",
)
model.fit(X,y)
f = model.get_best()['lambda_format']  # Callable
```

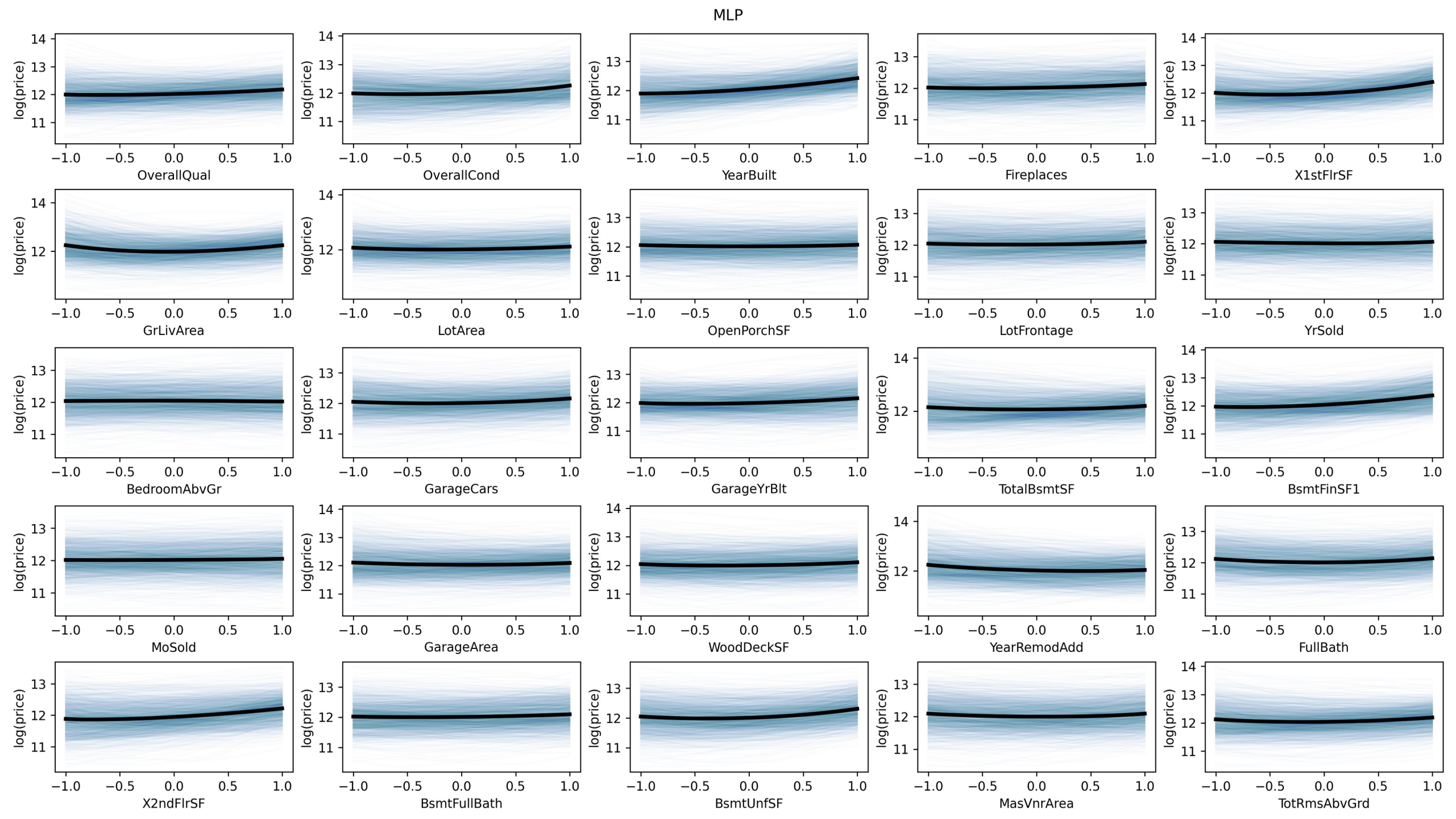
Interpretable machine learning for science with PySR and SymbolicRegression.jl, M. Cranmer (2023)

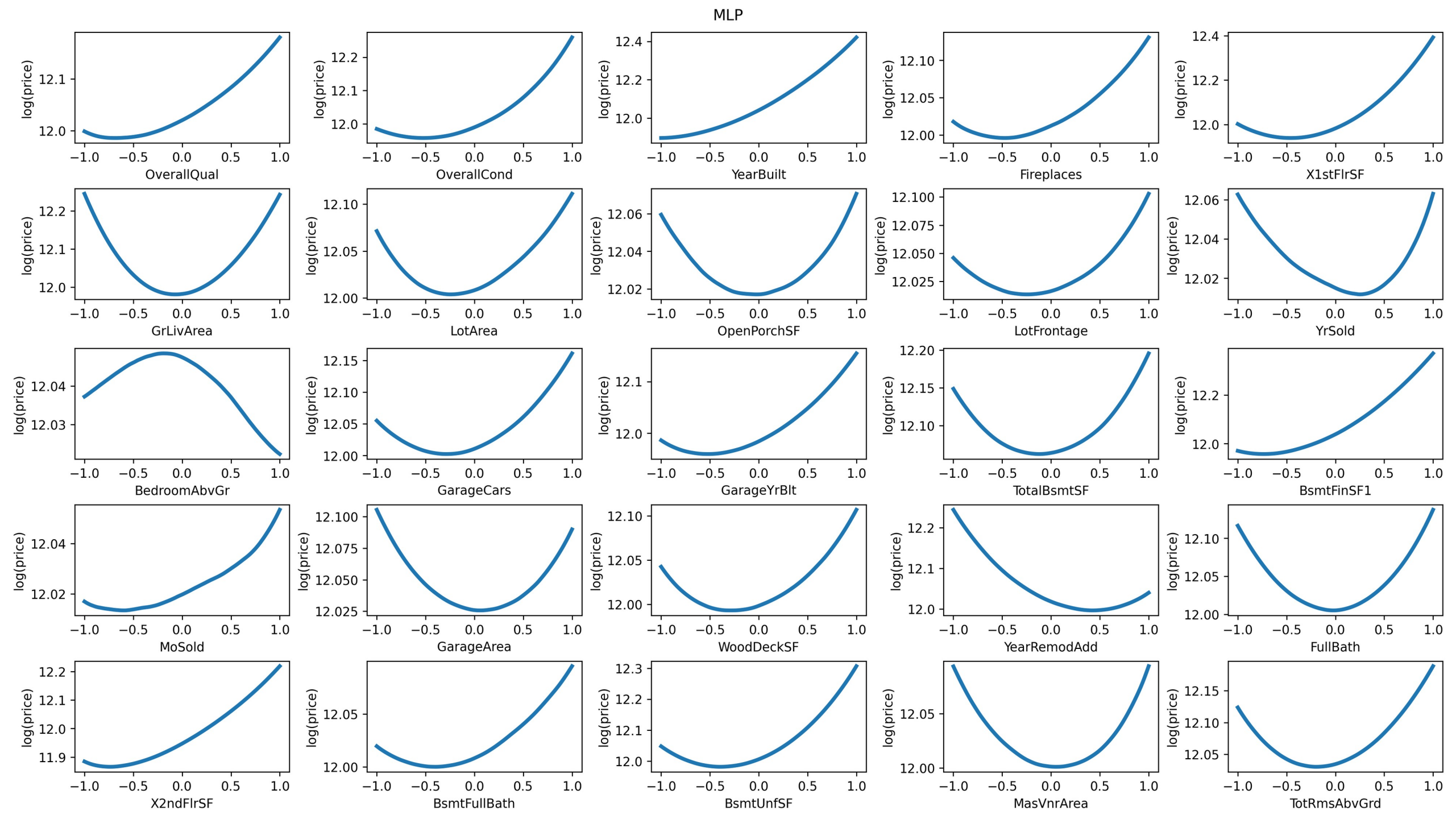
Deep Learning

MLP

```
class Net(nn.Module):
    def __init__(self, n_input, k1, k2):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(n_input, k1)
        self.fc2 = nn.Linear(k1, k2)
        self.fc3 = nn.Linear(k2, 1)
    def forward(self, x):
        h1 = F.relu( self.fc1(x) )
        h2 = F.relu( self.fc2(h1) )
        return self.fc3(h2)

model = Net( X.shape[1], 200, 200 )
optimizer = optim.Adam( model.parameters() )
loss_fn = nn.MSELoss(reduction='mean')
for epoch in range(10_000):
    optimizer.zero_grad()
    y_hat = model(X)
    loss = loss_fn( y_hat, y )
    loss.backward()
    optimizer.step()
```




MLP
log(price)
OverallQual
OverallCond
YearBuilt
Fireplaces
X1stFlrSF
GrLivArea
LotArea
OpenPorchSF
LotFrontage
YrSold
BedroomAbvGr
GarageCars
GarageYrBlt
TotalBsmtSF
BsmtFinSF1
MoSold
GarageArea
WoodDeckSF
YearRemodAdd
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF
MasVnrArea
TotRmsAbvGrd


MLP
log(price)
OverallQual
OverallCond
YearBuilt
Fireplaces
X1stFlrSF
GrLivArea
LotArea
OpenPorchSF
LotFrontage
YrSold
BedroomAbvGr
GarageCars
GarageYrBlt
TotalBsmtSF
BsmtFinSF1
MoSold
GarageArea
WoodDeckSF
YearRemodAdd
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF
MasVnrArea
TotRmsAbvGrd

Sparse-input neural nets

- Add a group lasso penalty to the first weight matrix

Ridge penalty for the other layers

group sparsity

sparsity

$$\text{loss} = \text{error} + \lambda_2 \sum_{a \geqslant 2} \|w_a\|_2^2 + \lambda_{12} \|w_1\|_{1,2} + \lambda_1 \|w_1\|_1$$

Sum of the L^2 norms of the columns

Sparse-input neural networks for high-dimensional nonparametric regression (J. Feng and N. Simon, 2017)

Sparse-input neural nets

- Standard PyTorch optimizers do no always lead to sparse solutions: separate the smooth terms from the sparsifying ones, and use the proximal operator for the latter

$$w \leftarrow w - \gamma \nabla_w \text{loss}_{\text{smooth}}$$

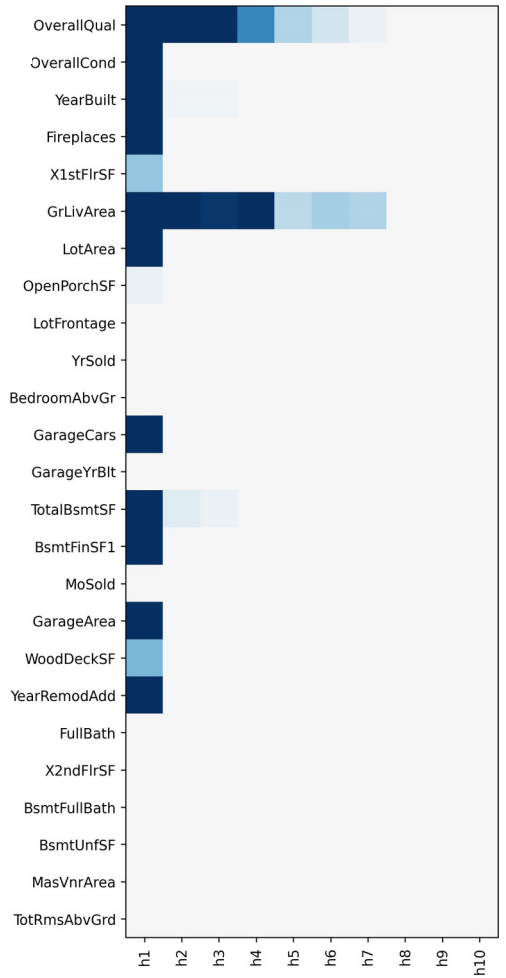
$$w_1 \leftarrow S(w_1, \gamma\lambda_1) \qquad \text{(soft-thresholding)}$$

$$w_{1,\cdot,i} \leftarrow \left(1 - \frac{\gamma\lambda_{12}}{\|w_{1,\cdot,i}\|_2}\right)_+ w_{1,\cdot,i}$$

Sparse-input neural networks for high-dimensional nonparametric regression (J. Feng and N. Simon, 2017)

Sparse-input neural nets

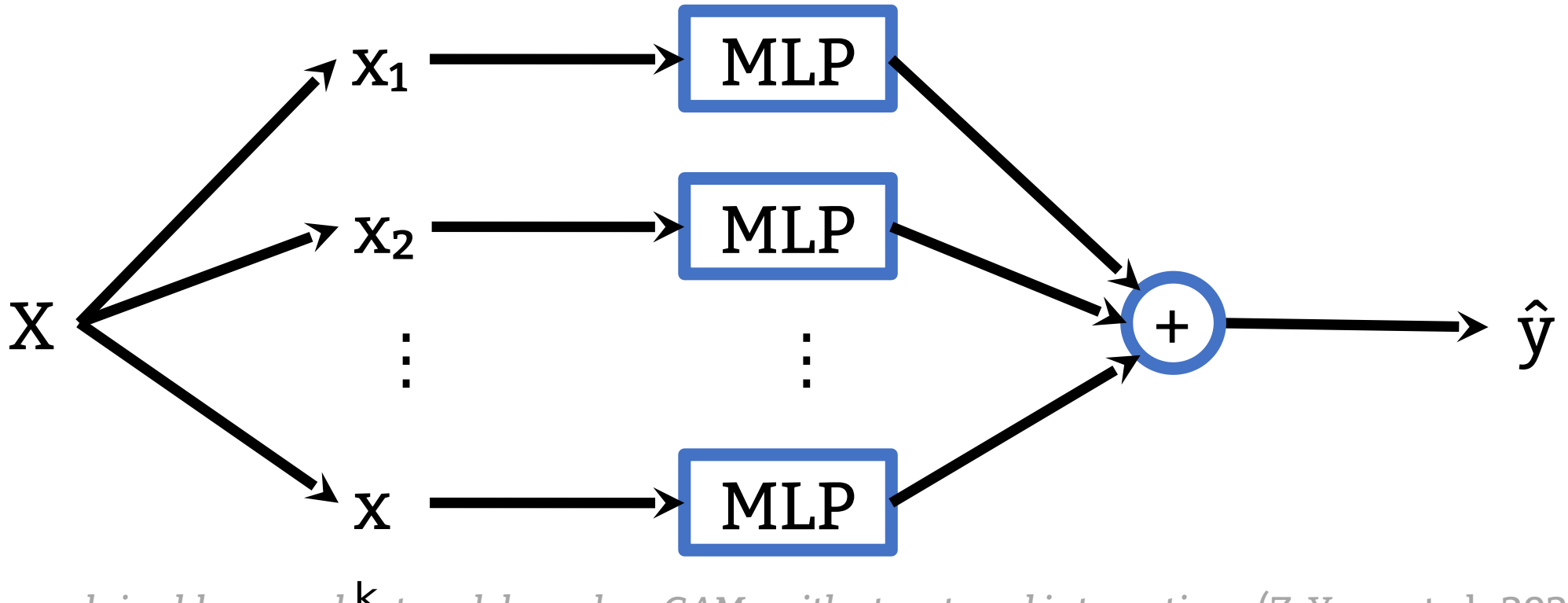
```
error = loss_fn( y_hat, y_ )
penalty_12 = torch.linalg.vector_norm( model.fc1.weight, ord=2, dim=0 ).sum()
penalty_1  = torch.linalg.vector_norm( model.fc1.weight, ord=1, dim=0 ).sum()
penalty_2  = (
    torch.linalg.vector_norm( model.fc2.weight, ord=2, dim=None ) ** 2 +
    torch.linalg.vector_norm( model.fc3.weight, ord=2, dim=None ) ** 2
)
loss_smooth = error + lambda_2 * penalty_2
loss_smooth.backward()
optimizer.step()

with torch.no_grad():
    soft_threshold_( model.fc1.weight, lr * lambda_1 * (1-alpha) )
    for i in range( model.fc1.weight.shape[1] ):
        w = model.fc1.weight[:,i]
        n = torch.linalg.vector_norm(w, ord=2)
        w.copy_( F.relu( 1 - lr * lambda_1 * alpha / n ) * w )
```



Sparse-input neural networks for high-dimensional nonparametric regression (J. Feng and N. Simon, 2017)

GAMI-Net

- GAM, implemented as a neural net, with sparsity penalty, and interactions.



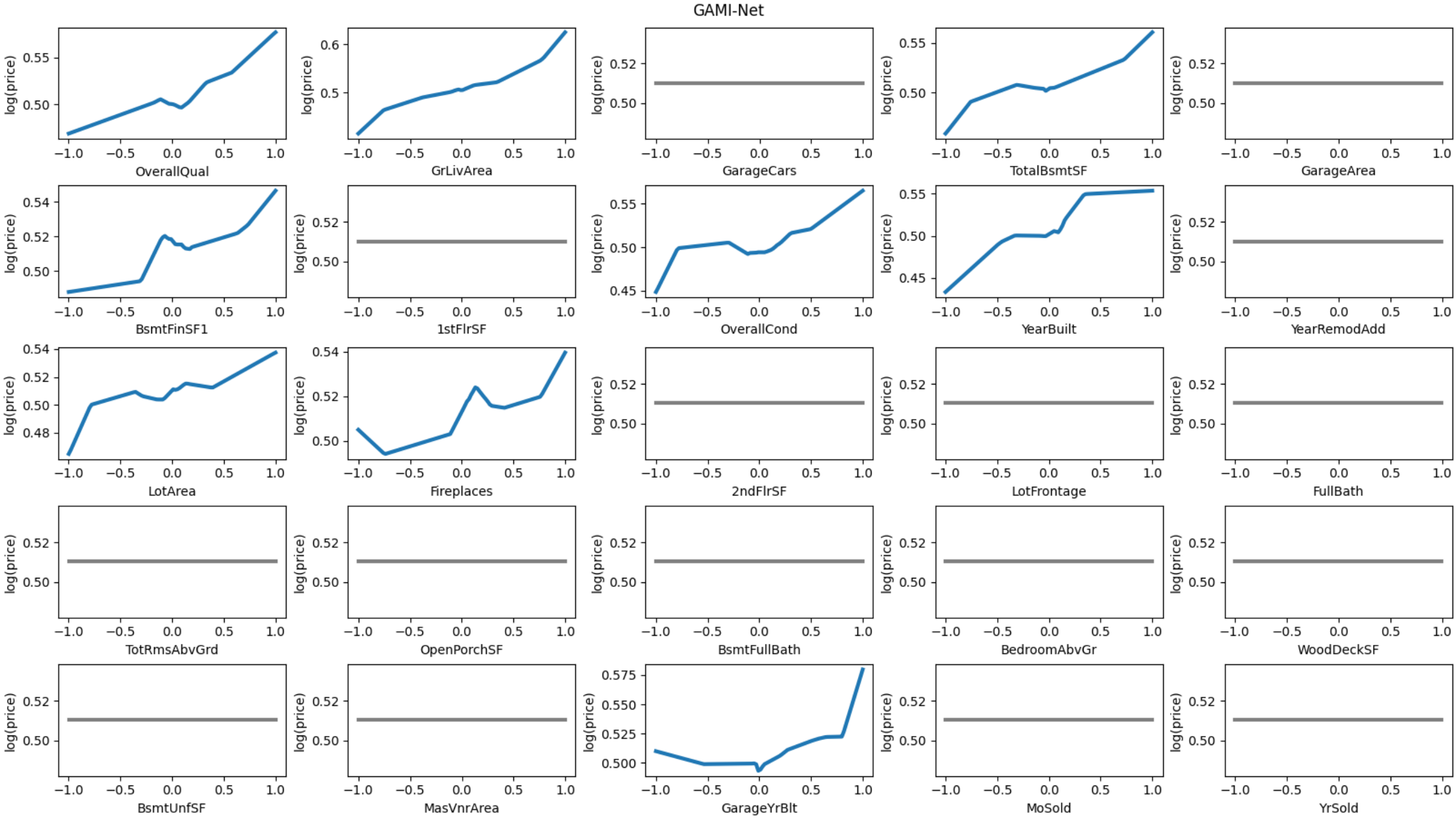
GAMI-Net: an explainable neural network based on GAMs with structured interactions (Z. Yang et al., 2020)

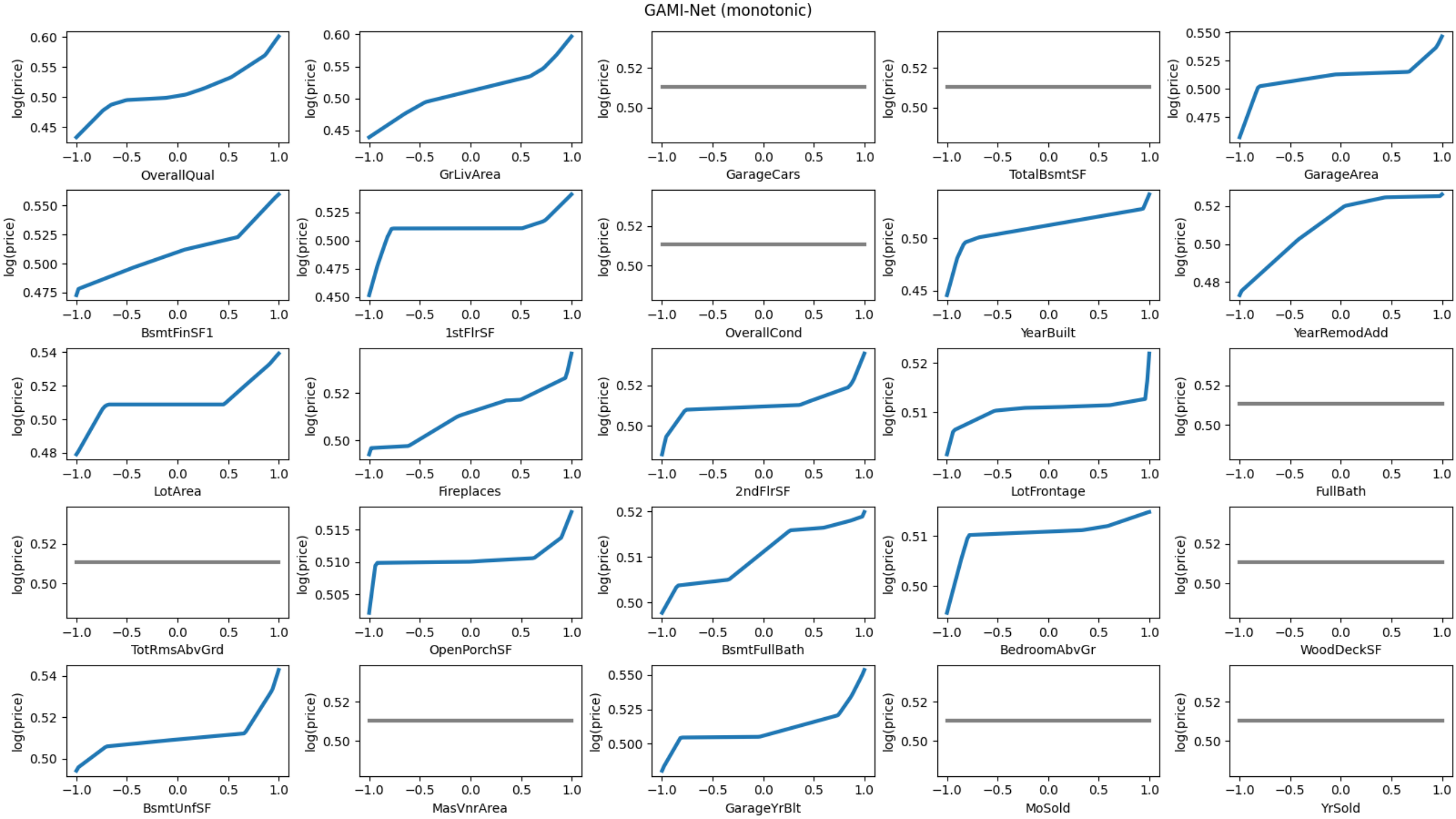
GAMI-Net

- GAM, implemented as a neural net, with sparsity penalty, and interactions.

```
from piml.models import GAMINetRegressor
from piml import Experiment

exp = Experiment()
exp.data_loader( data = pd.concat( [ X, y ], axis=1 ) )
exp.data_prepare(target=y.name, task_type='regression', sample_weight=None)
exp.model_train(model=GAMINetRegressor(mono_increasing_list=X.columns), name="GAMI-Net")
model = exp.get_model("GAMI-Net")
```

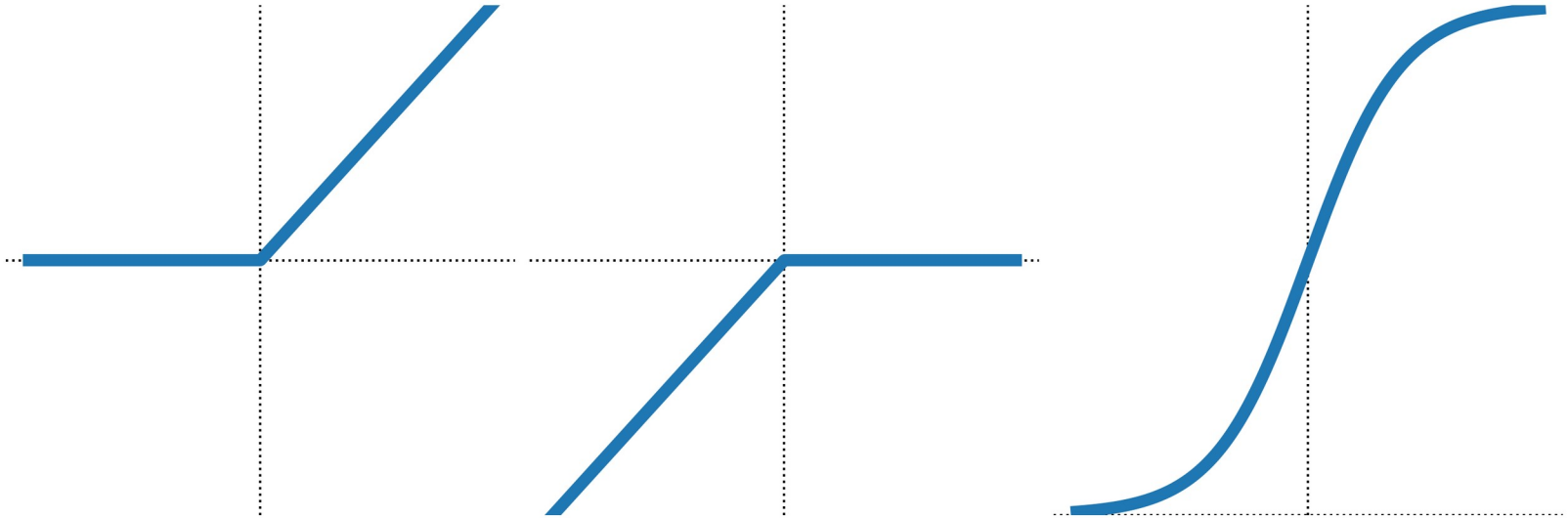
GAMI-Net: an explainable neural network based on GAMs with structured interactions (Z. Yang et al., 2020)


GAMI-Net
log(price)
OverallQual
GrLivArea
GarageCars
TotalBsmtSF
GarageArea
BsmtFinSF1
1stFlrSF
OverallCond
YearBuilt
YearRemodAdd
LotArea
Fireplaces
2ndFlrSF
LotFrontage
FullBath
TotRmsAbvGrd
OpenPorchSF
BsmtFullBath
BedroomAbvGr
WoodDeckSF
BsmtUnfSF
MasVnrArea
GarageYrBlt
MoSold
YrSold


GAMI-Net (monotonic)
log(price)
OverallQual
GrLivArea
GarageCars
TotalBsmtSF
GarageArea
BsmtFinSF1
1stFlrSF
OverallCond
YearBuilt
YearRemodAdd
LotArea
Fireplaces
2ndFlrSF
LotFrontage
FullBath
TotRmsAbvGrd
OpenPorchSF
BsmtFullBath
BedroomAbvGr
WoodDeckSF
BsmtUnfSF
MasVnrArea
GarageYrBlt
MoSold
YrSold

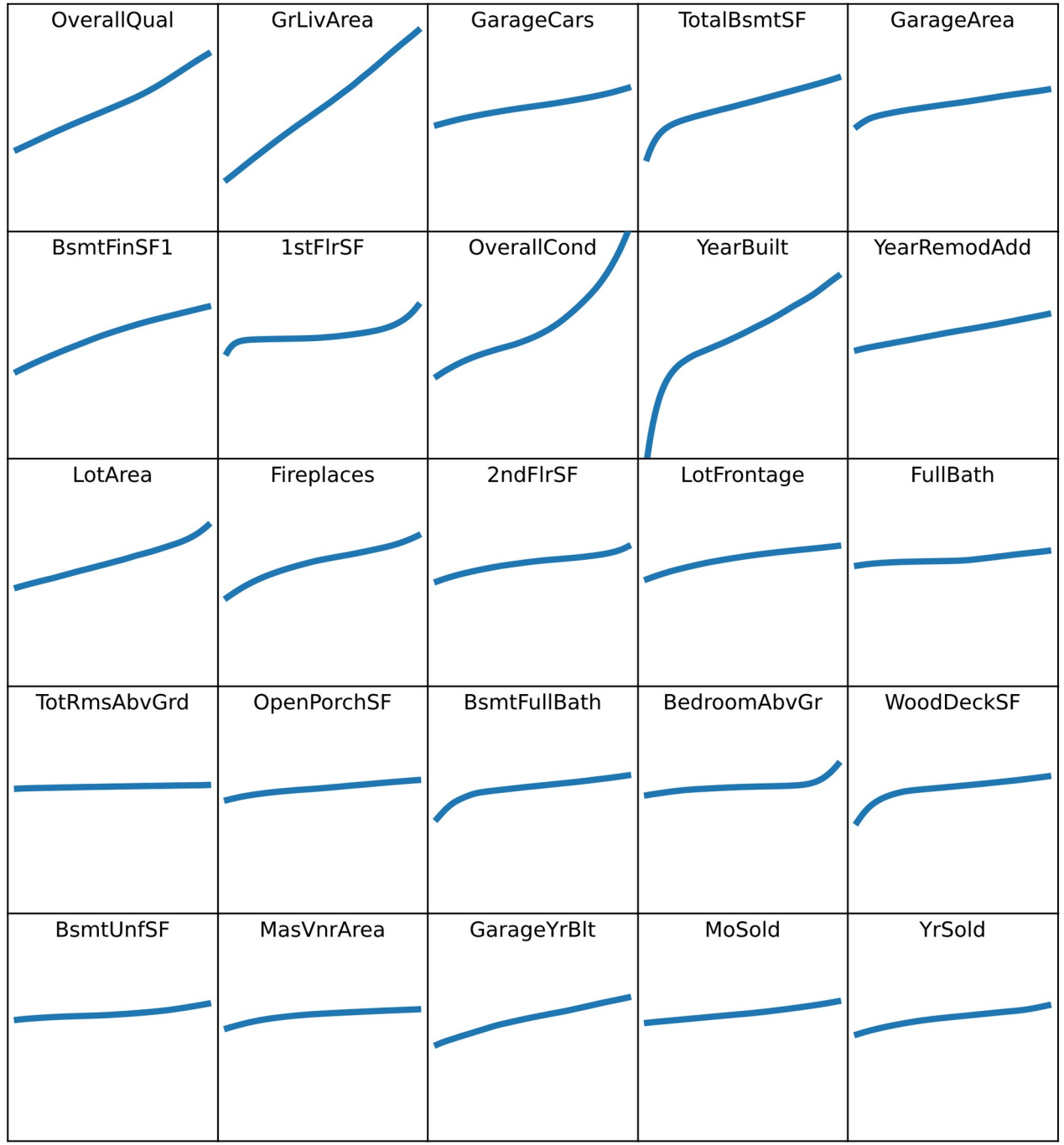
Monotonic MLP

- MLP with non-negative weights and ReLU activations: monotonic but convex
- Use non-negative weights and 3 activation functions



Constrained monotonic neural networks, D. Runje and S.M. Shankaranarayana (2022)

Three


OverallQual
GrLivArea
GarageCars
TotalBsmtSF
GarageArea
BsmtFinSF1
1stFlrSF
OverallCond
YearBuilt
YearRemodAdd
LotArea
Fireplaces
2ndFlrSF
LotFrontage
FullBath
TotRmsAbvGrd
OpenPorchSF
BsmtFullBath
BedroomAbvGr
WoodDeckSF
BsmtUnfSF
MasVnrArea
GarageYrBlt
MoSold
YrSold

Gradient penalty

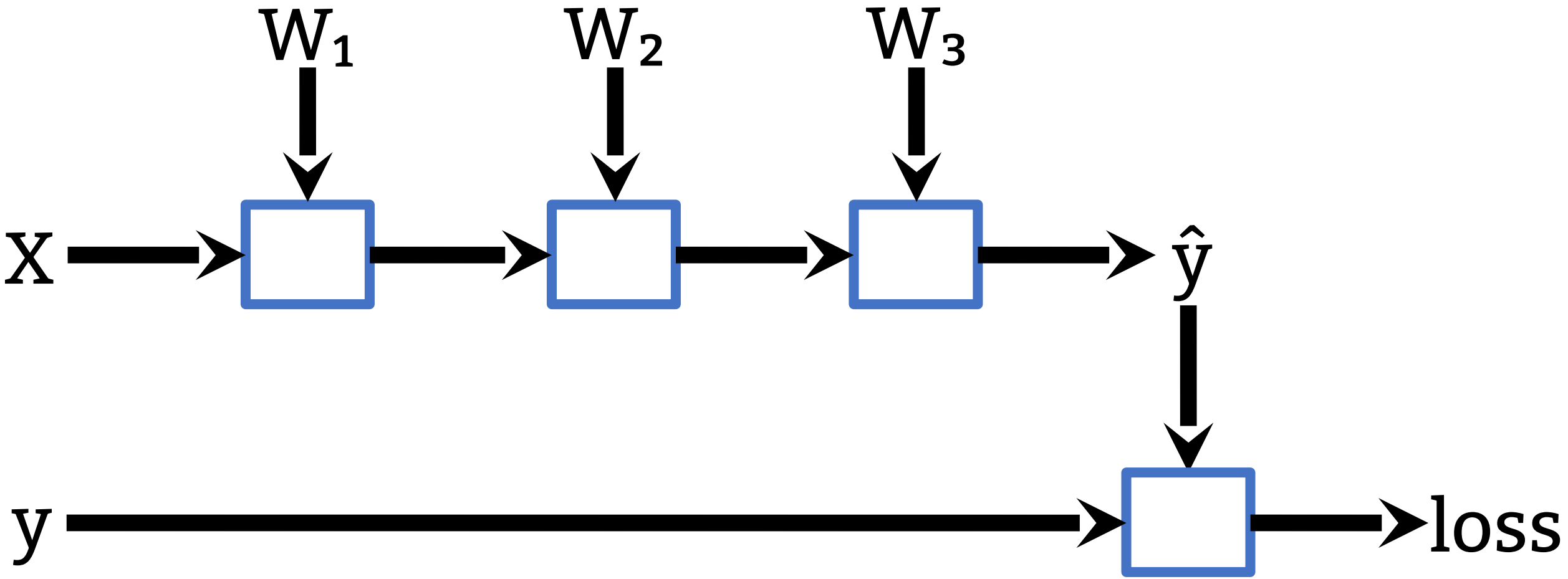
Add a penalty when the gradient ∂ŷ/∂X is negative.

Notes:

- This is not the gradient used to train the network, ∂loss/∂W
- The gradient computations have to be added to the computation graph
- This is a penalty, not a hard constraint

How to incorporate monotonicity in deep networks while preserving flexibility? A. Gupta et al. (2019)

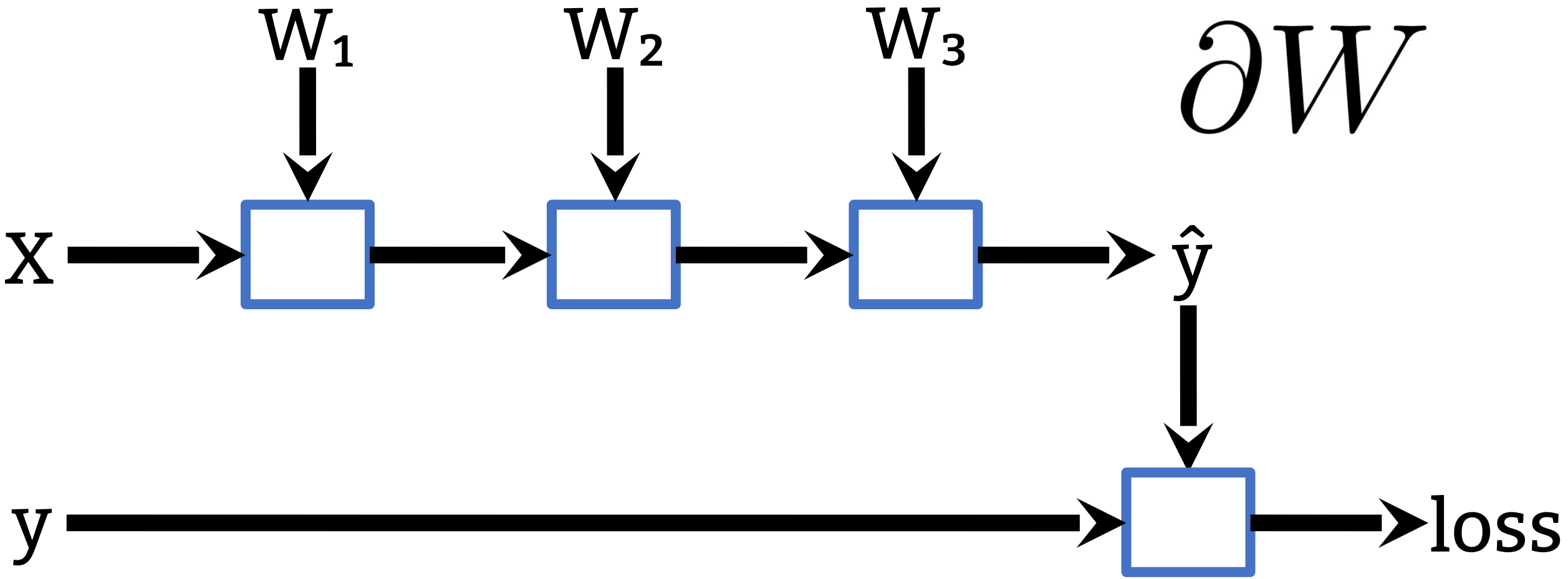
Gradient penalty



How to incorporate monotonicity in deep networks while preserving flexibility? A. Gupta et al. (2019)

Gradient penalty

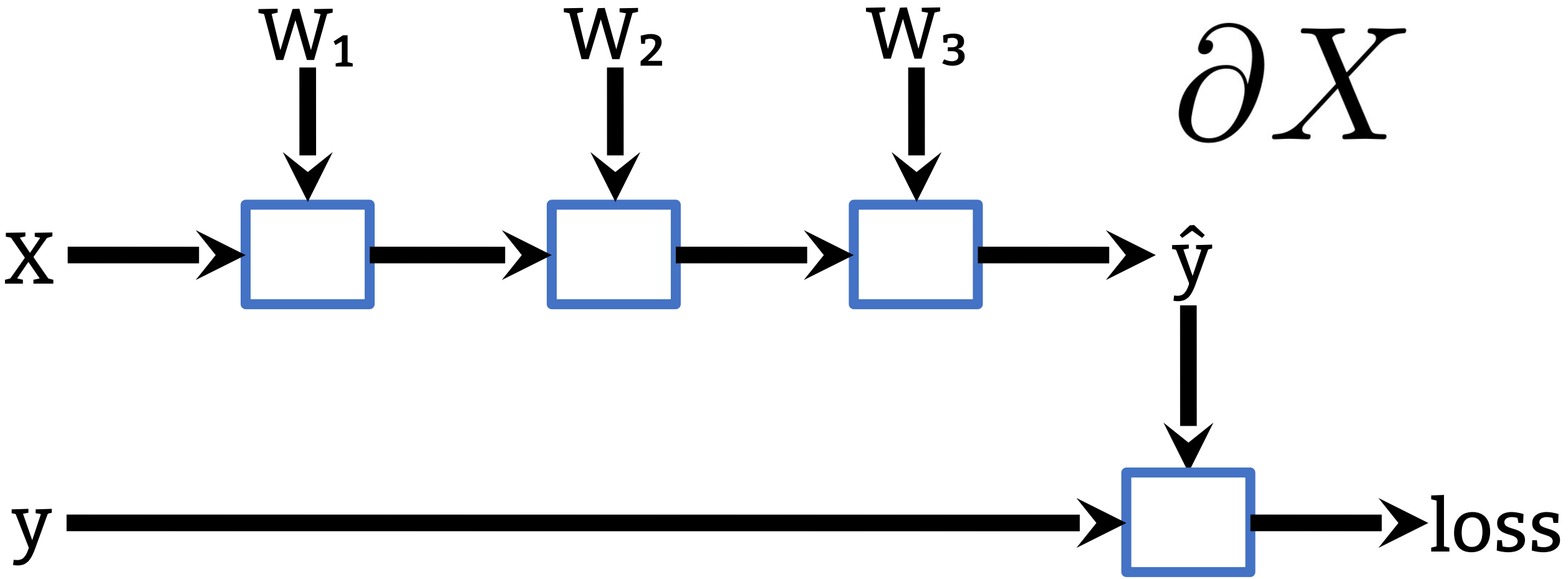
$$\frac{\partial \text{loss}}{\partial W}$$



How to incorporate monotonicity in deep networks while preserving flexibility? A. Gupta et al. (2019)

Gradient penalty

$$\frac{\partial \hat{y}}{\partial X}$$



How to incorporate monotonicity in deep networks while preserving flexibility? A. Gupta et al. (2019)

Code: gradient penalty

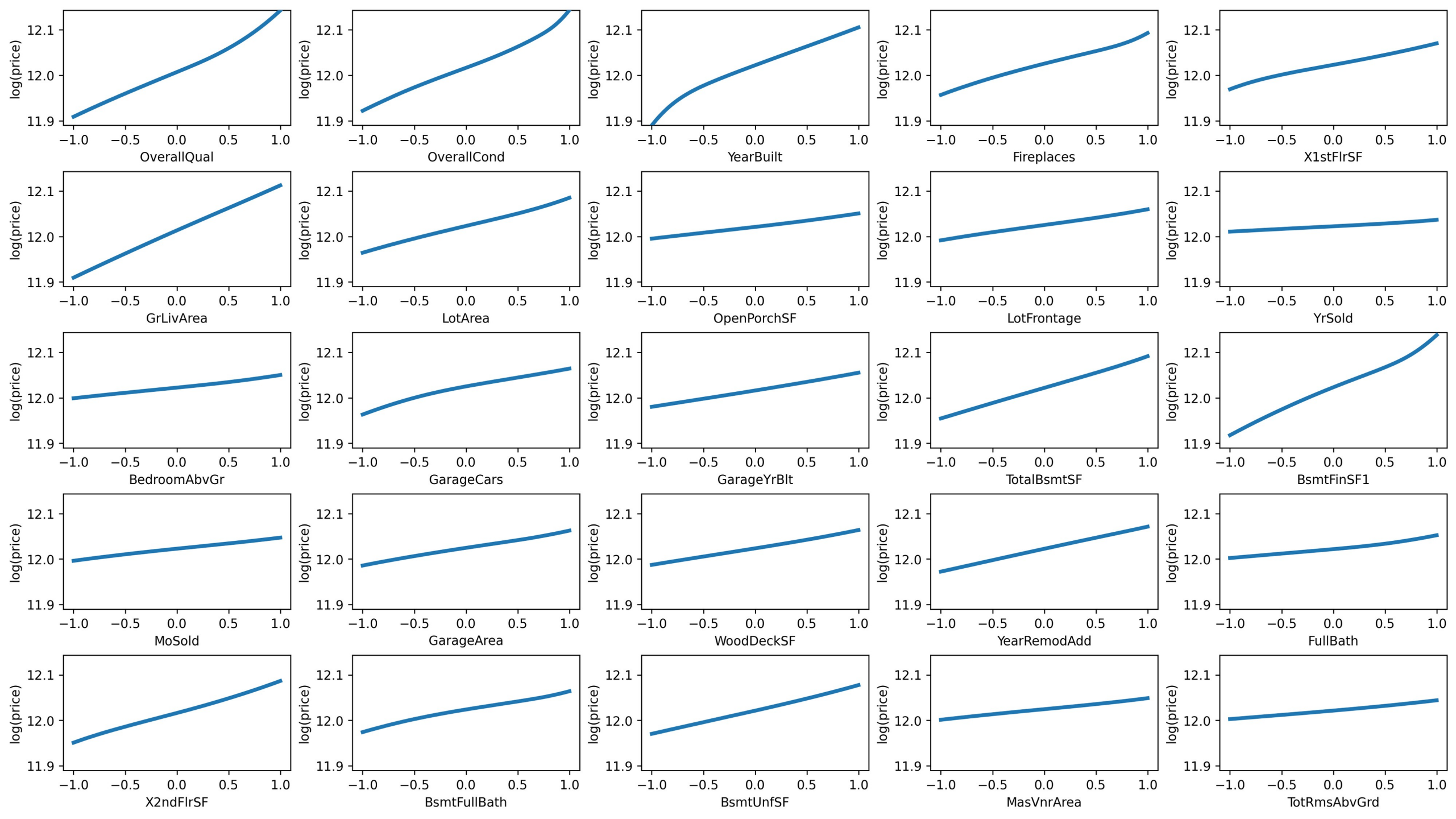
```
# Gradient penalty
for epoch in range(10_000):
    optimizer.zero_grad()
    y_hat = model(X)
    error = loss_fn( y_hat, y )
    g = torch.autograd.grad(
        outputs = y_hat.sum(),
        inputs  = X,
        grad_outputs = torch.ones(y_hat.size()),
        create_graph = True,
        retain_graph = True,
    )[0]
    # Hinge loss, progressively increasing
    penalty = torch.relu(-g+.001).mean() * epoch/100
    loss = error + penalty
    loss.backward()
    optimizer.step()
```

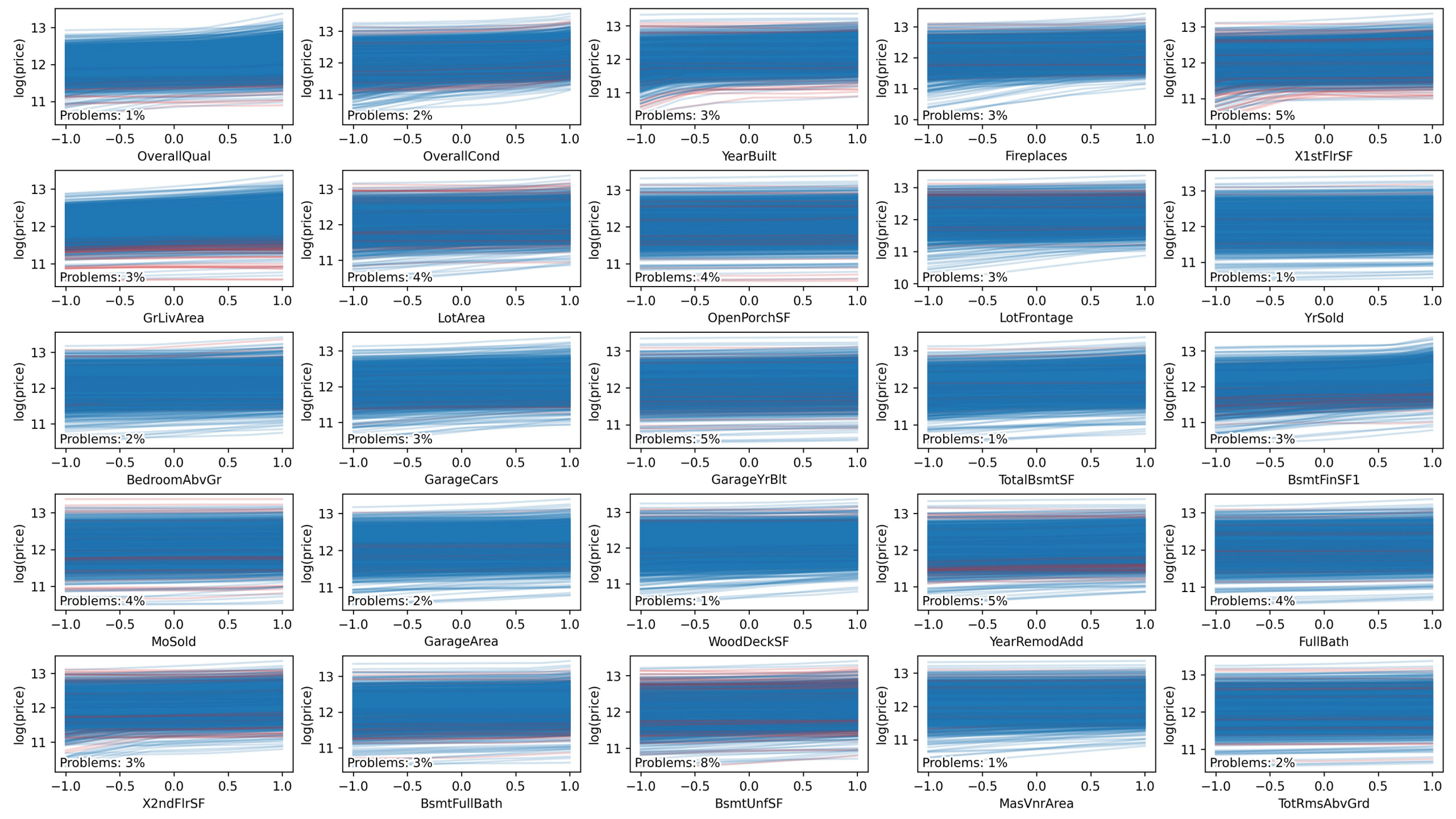
Code: gradient penalty

```
# Gradient penalty
for epoch in range(10_000):
    optimizer.zero_grad()
    y_hat = model(X)
    error = loss_fn( y_hat, y )
    g = torch.autograd.grad(
        outputs = y_hat.sum(),
        inputs  = X,
        grad_outputs = torch.ones(y_hat.size()),
        create_graph = True,
        retain_graph = True,
    )[0]
    # Hinge loss, progressively increasing
    penalty = torch.relu(-g+.001).mean() * epoch/100
    loss = error + penalty
    loss.backward()
    optimizer.step()
```

Code: gradient penalty

```
# Gradient penalty
for epoch in range(10_000):
    optimizer.zero_grad()
    y_hat = model(X)
    error = loss_fn( y_hat, y )
    g = torch.autograd.grad(
        outputs = y_hat.sum(),
        inputs  = X,
        grad_outputs = torch.ones(y_hat.size()),
        create_graph = True,
        retain_graph = True,
    )[0]
    # Hinge loss, progressively increasing
    penalty = torch.relu(-g+.001).mean() * epoch/100
    loss = error + penalty
    loss.backward()
    optimizer.step()
```


log(price)
12.1
12.0
11.9
−1.0
−0.5
0.0
0.5
1.0
OverallQual
OverallCond
YearBuilt
Fireplaces
X1stFlrSF
GrLivArea
LotArea
OpenPorchSF
LotFrontage
YrSold
BedroomAbvGr
GarageCars
GarageYrBlt
TotalBsmtSF
BsmtFinSF1
MoSold
GarageArea
WoodDeckSF
YearRemodAdd
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF
MasVnrArea
TotRmsAbvGrd


log(price)
Problems: 1%
OverallQual
Problems: 2%
OverallCond
Problems: 3%
YearBuilt
Problems: 3%
Fireplaces
Problems: 5%
X1stFlrSF
Problems: 3%
GrLivArea
Problems: 4%
LotArea
Problems: 4%
OpenPorchSF
Problems: 3%
LotFrontage
Problems: 1%
YrSold
Problems: 2%
BedroomAbvGr
Problems: 3%
GarageCars
Problems: 5%
GarageYrBlt
Problems: 1%
TotalBsmtSF
Problems: 3%
BsmtFinSF1
Problems: 4%
MoSold
Problems: 2%
GarageArea
Problems: 1%
WoodDeckSF
Problems: 5%
YearRemodAdd
Problems: 4%
FullBath
Problems: 3%
X2ndFlrSF
Problems: 3%
BsmtFullBath
Problems: 8%
BsmtUnfSF
Problems: 1%
MasVnrArea
Problems: 2%
TotRmsAbvGrd

MIP monotonicity proof

- A neural network with linear layers and ReLU activations is a piecewise linear function: it can be used in a linear program.
- Monotonicity can be formulated as the infeasibility of a linear program.

$$
\begin{array}{ll}
\text{Find} & x_1, x_2 \in \mathbf{R}^d \\
\text{Such that} & \forall i \; x_{1i} \leqslant x_{2i} \\
& f(x_1) > f(x_2)
\end{array}
$$

Certified monotonic neural networks, X. Liu et al. (2020)

MIP monotonicity proof

- A neural network with linear layers and ReLU activations is a piecewise linear function: it can be used in a linear program.
- Monotonicity can be formulated as the infeasibility of a linear program.

$$
\begin{array}{ll}
\text{Find} & x_1, x_2 \in \mathbf{R}^d \\
\text{Such that} & \forall i \; x_{1i} \leqslant x_{2i} \\
& f(x_1) \geqslant f(x_2) + \varepsilon
\end{array}
$$

Certified monotonic neural networks, X. Liu et al. (2020)

MIP monotonicity proof

- A neural network with linear layers and ReLU activations is a piecewise linear function: it can be used in a linear program.
- Monotonicity can be formulated as the infeasibility of a linear program.

$$\begin{array}{ll} \text{Find} & x_1, x_2 \in \mathbf{R}^d \\ \text{To maximize} & f(x_1) - f(x_2) \\ \text{Such that} & \forall i \; x_{1i} \leqslant x_{2i} \end{array}$$

Certified monotonic neural networks, X. Liu et al. (2020)

Adversarial examples

Use such counter-examples during training:

- Every other iteration, for each sample x, find the worst counter-examples in each direction, x_l, x_u, and replace y with the average of $f(x_l)$ and $f(x_u)$

Counterexample-guided learning of monotonic neural networks, A. Sivaraman et al. (2020)

Varying coefficient models

Locally linear model:

$$f(x) = \sum_i \theta_i(x) x_i$$

where θ varies very slowly.

Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

Varying coefficient models

Locally linear model:

$$f(x) = \sum_i \theta_i(x) x_i$$

where θ varies very slowly.

Often, θ only depends on a small number (one) of the x_i's.

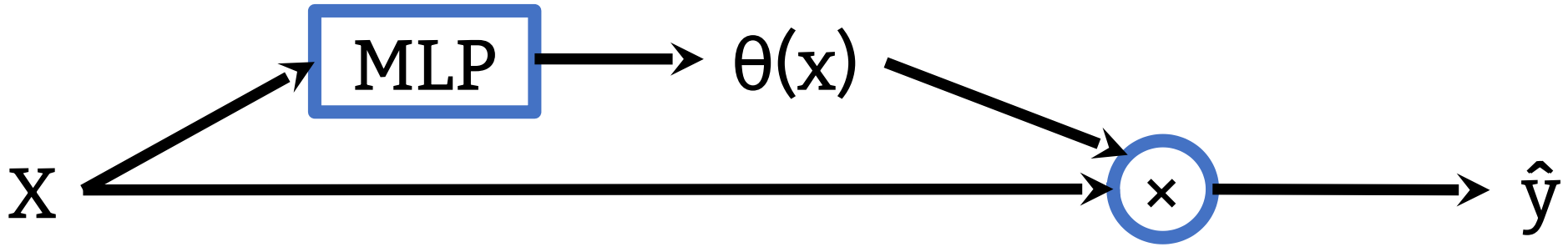
Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

Varying coefficient models

$$\text{Reconstruction loss: } \|x - \hat{x}\|^2$$

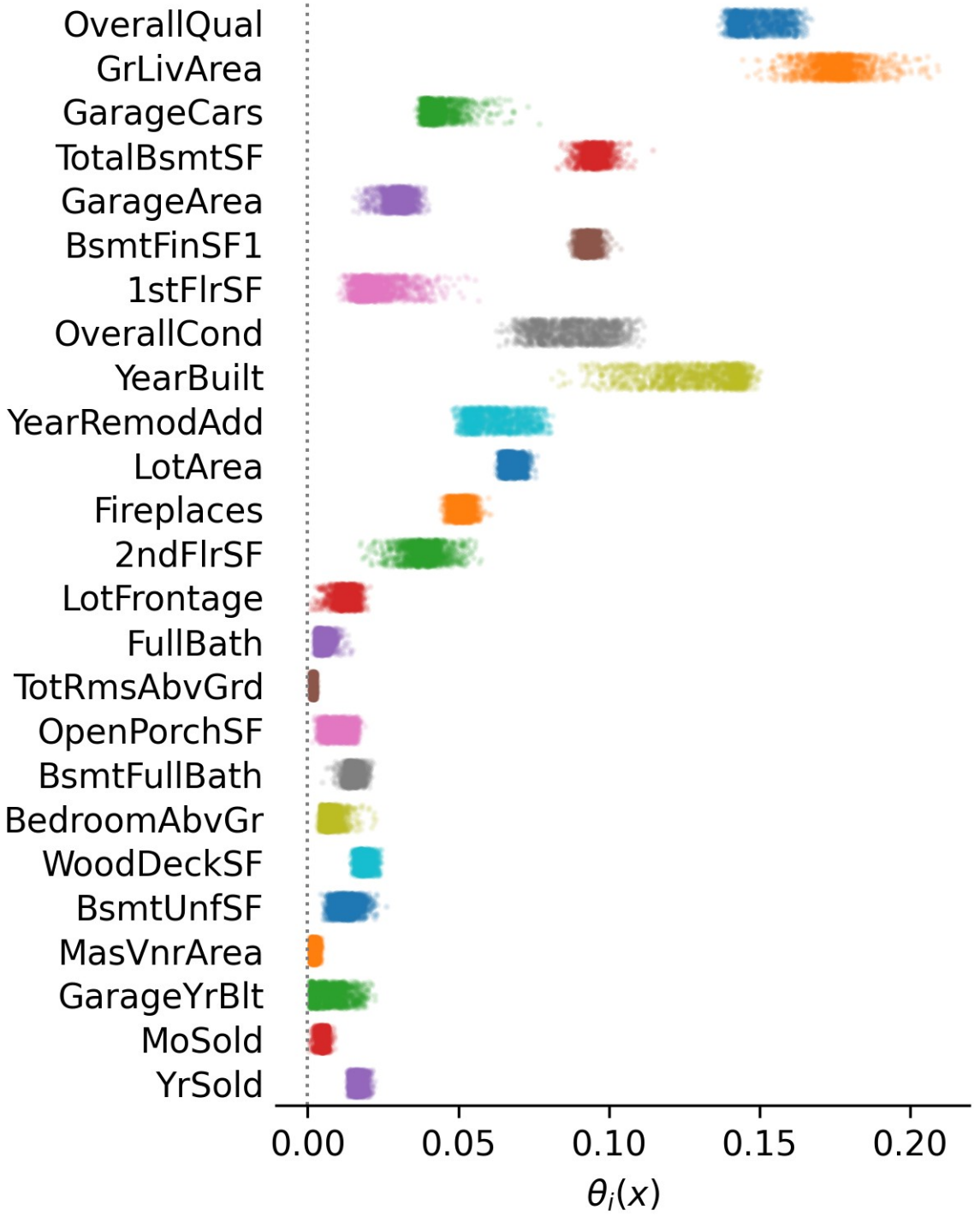
$$\text{Prediction loss: } \|y - \hat{y}\|^2$$

$$\text{Penalty: } \left\| \frac{\partial \hat{y}}{\partial x} - \theta \right\|^2$$



Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

```python
class Net(nn.Module):  # VCM
    def __init__(self, n_inputs, k):
        super(Net, self).__init__()
        self.fc1 = nn.Linear( n_inputs, k )
        self.fc2 = nn.Linear( k, n_inputs )
        self.intercept = nn.Parameter( torch.tensor(0.) )
    def forward(self, x):
        h = F.relu( self.fc1(x) )
        beta = self.fc2(h)
        y = self.intercept + torch.sum( beta * x, dim=1, keepdim=True )
        return beta, y

model = Net( X_.shape[1], 20 )
with torch.no_grad():
    model.intercept.copy_( torch.tensor( y.mean() ) )

optimizer = optim.AdamW( model.parameters() )
loss_fn = nn.MSELoss(reduction='mean')
for epoch in tqdm(range(10_000)):
    optimizer.zero_grad()
    beta, y_hat = model(X_)
    error = loss_fn( y_hat, y_ )
    penalty = torch.tensor(0)
    g = torch.autograd.grad(
        outputs = y_hat,
        inputs = X_,
        grad_outputs = torch.ones(y_hat.size()),
        create_graph = True,
        retain_graph = True,
    )[0]
    penalty = torch.sum( ( g - beta ) ** 2 ) / X_.shape[0]
    penalty_pos = F.relu( -beta ).sum()
    loss = error + 10 * penalty + 1e-2 * penalty_pos
    loss.backward()
    optimizer.step()
```



Self-explaining neural nets

Locally linear model:

$$f(x) = \sum_i \theta_i(x) h_i(x)$$

where θ varies very slowly and *h* comes from an auto-encoder.

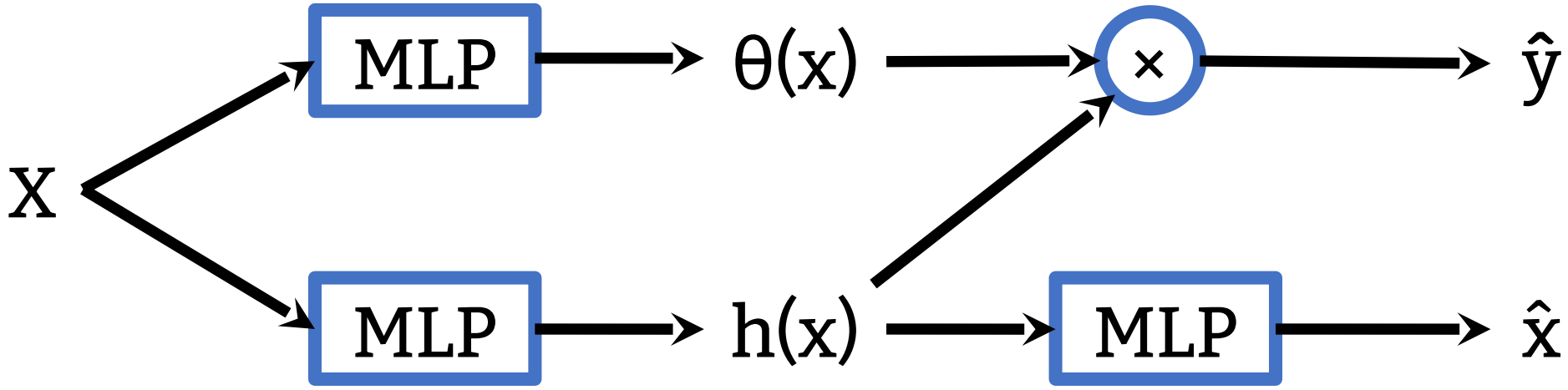
Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

Self-explaining neural nets

$$\text{Reconstruction loss: } \|x - \hat{x}\|^2$$

$$\text{Prediction loss: } \|y - \hat{y}\|^2$$

$$\text{Penalty: } \|\nabla_x \hat{y} - \theta \nabla_x h\|^2$$



Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

```
class Net(nn.Module):
    def __init__(self, n_inputs, theta_k, h_k1, h_k2):
        super(Net, self).__init__()
        self.theta_fc1 = nn.Linear( n_inputs, theta_k )
        self.theta_fc2 = nn.Linear( theta_k, h_k2 )
        self.intercept = nn.Parameter( torch.tensor(0.) )
        self.enc_fc1 = nn.Linear( n_inputs, h_k1 )
        self.enc_fc2 = nn.Linear( h_k1, h_k2 )
        self.dec_fc1 = nn.Linear( h_k2, h_k1 )
        self.dec_fc2 = nn.Linear( h_k1, n_inputs )
    def forward(self, x):
        h    = self.enc_fc2( F.relu( self.enc_fc1( x ) ) )
        xhat = self.dec_fc2( F.relu( self.dec_fc1( h ) ) )
        theta = self.theta_fc2( F.relu( self.theta_fc1(x) ) )
        yhat = self.intercept + torch.sum( theta * h, dim=1, keepdim=True )
        return h, xhat, theta, yhat

h, x_hat, beta, y_hat = model(X_)
error          = loss_fn( y_hat, y_ )
reconstruction = loss_fn( x_hat, X_ )
g = torch.autograd.grad(
    outputs = y_hat,
    inputs = X_,
    grad_outputs = torch.ones(y_hat.size()),
    create_graph = True,
    retain_graph = True,
)[0]
Js = [  # Isn't there an easier way to do that?
    torch.autograd.grad(
        outputs = h[:,i],
        inputs  = X_,
        grad_outputs = torch.ones(h[:,i].size()),
        retain_graph = True,
        create_graph = True,
    )[0]
    for i in range(h.shape[1])
]
Js = [ J.view( J.shape[0], J.shape[1], 1 ) for J in Js ]
J = torch.concat(Js,2)
penalty = torch.sum( ( g - torch.einsum('ik,ijk->ij', beta, J) ) ** 2 ) / X_.shape[0]
loss = error + reconstruction + 10 * penalty
```

Mixture of experts

$$\text{GAM} \qquad f(x) = \sum_i f_i(x_i)$$

$$\text{VCM} \qquad f(x) = \sum_i \beta_i(x) x_i$$

$$\text{AME} \qquad f(x) = \sum_i \beta_i(x) f_i(x_i)$$

VCM GAM

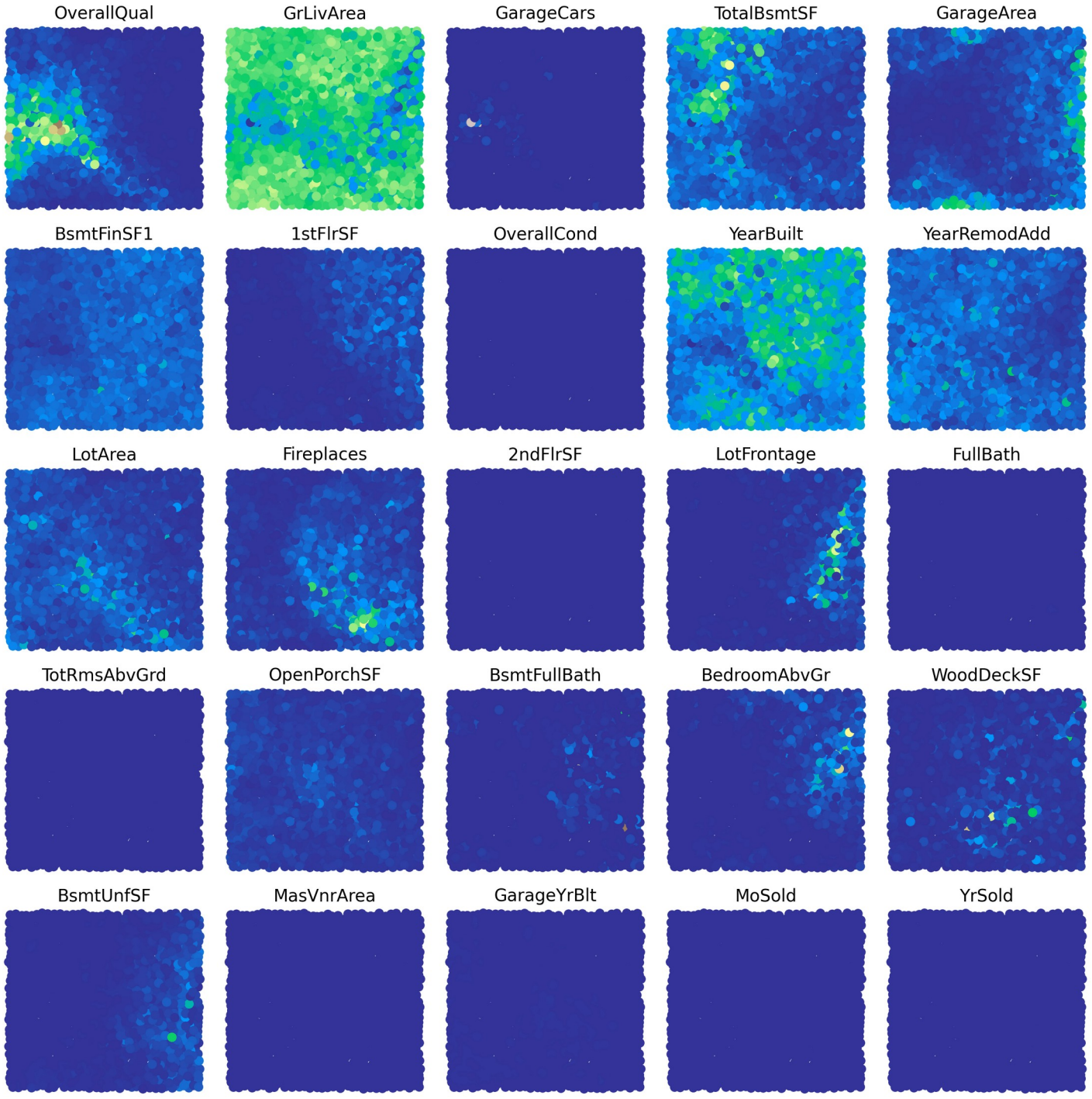
Granger-causal attentive mixtures of experts (P. Schwab et al., 2018)

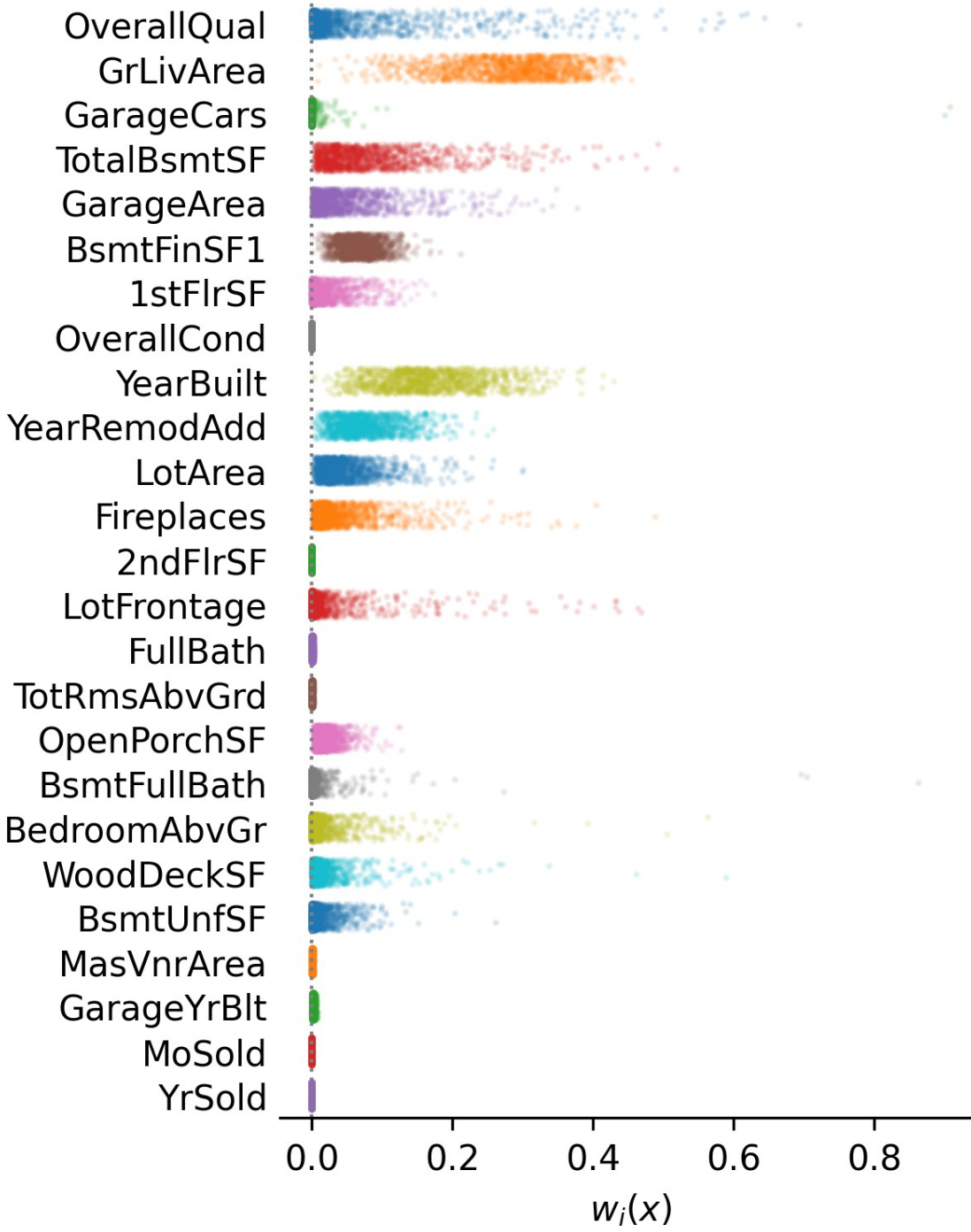
Mixture of experts

$$f(x) = \sum_i \beta_i(x) f_i(x_i)$$

- Penalty to make $f_i(x_i)$ a predictor of y
- Penalty to make the $f_i(x_i)$ have the same variance
- Penalty to make the f_i monotonic
- Penalty to make β almost constant
- Softmax for the weights: $\sum \beta_i(x) = 1$, $\beta_i \geqslant 0$
- Penalty to make the β_i reflect the drop in performance if we remove feature i

Granger-causal attentive mixtures of experts (P. Schwab et al., 2018)


OverallQual
GrLivArea
GarageCars
TotalBsmtSF
GarageArea
BsmtFinSF1
1stFlrSF
OverallCond
YearBuilt
YearRemodAdd
LotArea
Fireplaces
2ndFlrSF
LotFrontage
FullBath
TotRmsAbvGrd
OpenPorchSF
BsmtFullBath
BedroomAbvGr
WoodDeckSF
BsmtUnfSF
MasVnrArea
GarageYrBlt
MoSold
YrSold


OverallQual
GrLivArea
GarageCars
TotalBsmtSF
GarageArea
BsmtFinSF1
1stFlrSF
OverallCond
YearBuilt
YearRemodAdd
LotArea
Fireplaces
2ndFlrSF
LotFrontage
FullBath
TotRmsAbvGrd
OpenPorchSF
BsmtFullBath
BedroomAbvGr
WoodDeckSF
BsmtUnfSF
MasVnrArea
GarageYrBlt
MoSold
YrSold
0.0
0.2
0.4
0.6
0.8
$w_i(x)$

Conclusion

Conclusion

- Add sign constraints or **monotonicity** penalties to your models
- Add **sparsity** penalties to your models
- Prefer **additive** models (GAM, GA²M): while **sparse** and interpretable, they allow for nonlinearities and interactions

Test MSE

Model	MSE
Constant	0.150
OLS	0.022
Constrained regression	0.026
Decision tree	0.057
Monotonic DT	0.057
XGBoost	0.021
LightGBM	0.020
CatBoost	0.019
XGBoost (monotonic)	0.024
LightGBM (monotonic)	0.020
CatBoost (monotonic)	0.033
GAM	0.019
GAM (monotonic)	**0.017**

Model	MSE
Neural Net	0.058
AIM	0.040
Symbolic regression	0.033
GAM Boosting	0.025
NN with positive weights	0.028
NN with gradient penalty	0.022
Sparse-input NN	0.032
GAMI-Net	0.020
GAMI-Net (monotonic)	0.020
Varying coefficient model	0.023
Self-explaining neural net	0.021
Mixture of experts	0.018

References

Interpretable Machine Learning, C. Molnar (2020)

Why should I trust you? Explaining the predictions of any classifier, M.T. Ribeiro et al. (2016)

Designing inherently interpretable machine learning models, A. Sudjianto and A. Zhang (2021)

Statistical Learning with Sparsity, T. Hastie et al. (2015)

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

An optimization approach to learning falling rule lists, C. Chen and C. Rudin (2017)

Generalized additive models (S.N. Wood, 2017)

Model-based boosting in R: a hands-on tutorial using the R package mboost (B. Hofner et al., 2014)

Interpretable machine learning for science with PySR and SymbolicRegression.jl, M. Cranmer (2023)

GAMI-Net: an explainable neural network based on GAMs with structured interactions (Z. Yang et al., 2020)

Constrained monotonic neural networks, D. Runje and S.M. Shankaranarayana (2022)

How to incorporate monotonicity in deep networks while preserving flexibility? A. Gupta et al. (2019)

Certified monotonic neural networks, X. Liu et al. (2020)

Counterexample-guided learning of monotonic neural networks, A. Sivaraman et al. (2020)

Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

Interpretable and explainable machine learning: a methods-centric overview with concrete examples (R. Marcinkevičs and J.E. Vogt, 2023)

Granger-causal attentive mixtures of experts (P. Schwab et al., 2018)

Extra Slides

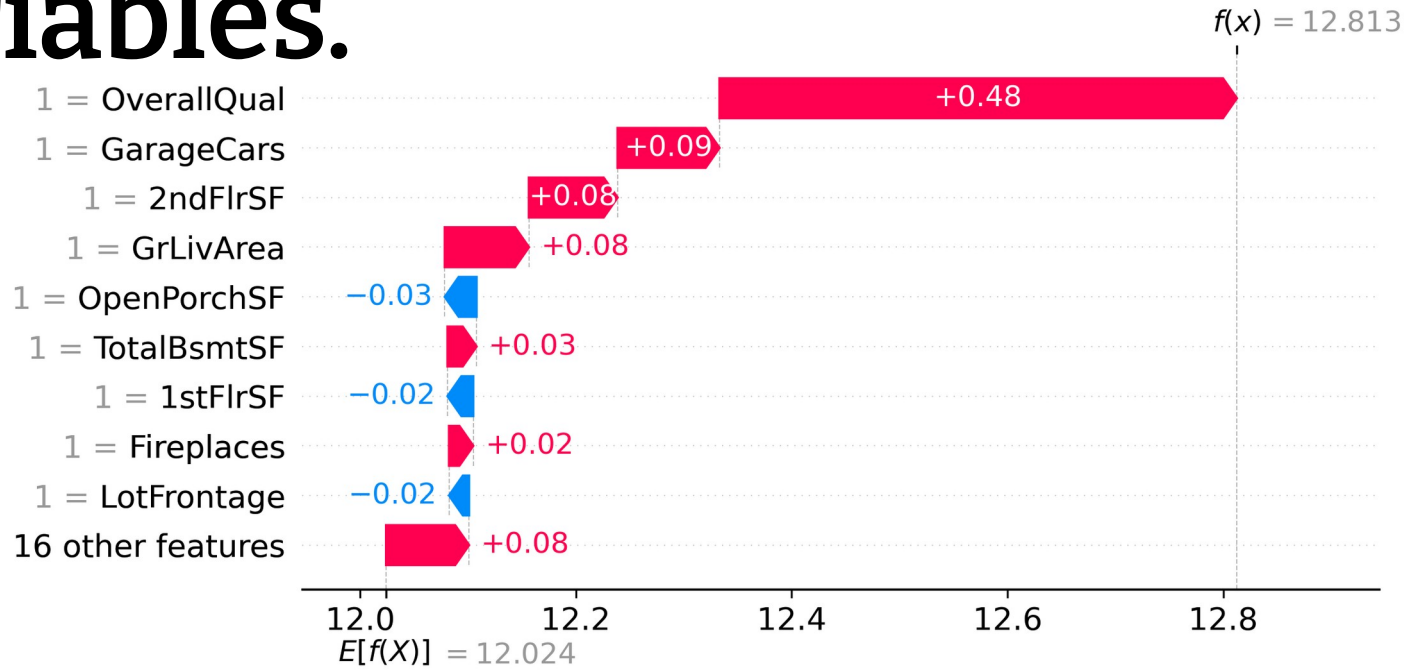
XAI

Shapley Values

- Decomposition of the performance of a model into a sum of contributions of each input variable, obtained by fitting the model on all subsets of variables.

$$A_I = \sum_{i \in I} A_i$$

$$A_i = \underset{S \ni i}{\text{Mean}}\, A_S - A_{S \setminus \{i\}}$$



Global Surrogate Models

- Simpler, interpretable model (e.g., linear regression or shallow regression tree), trained to reproduce the output of a more complicated model.

Interpretable Machine Learning, C. Molnar (2020)

Local Surrogate Models

- **Simpler, interpretable model (e.g., linear regression or shallow regression tree), trained to reproduce the output of a more complicated model, but only in the vicinity of a given observation.**

> *"For observations similar to this one, the model works as follows: …"*

Why should I trust you? Explaining the predictions of any classifier, M.T. Ribeiro et al. (2016)

Counterfactuals

- For a given observation, which input variable should we change to change the ouput?

Saliency maps, etc.

- For a given observation, decompose the output into a sum of contributions of each input variable.

Evaluating the visualization of what a deep neural network has learned, W. Samek et al. (2015)

SLIM

$$\text{Find} \quad \beta \in [\![-10, 10,]\!]^k$$

$$\text{To minimize} \quad \sum_i \text{loss}\left(y_i, \text{sign}(\beta' x_i)\right) + \lambda \|\beta\|_0 + \epsilon \|\beta\|_1$$

We want many of the β_i to be 0

0-1 loss

Binary output: ±1

Binary inputs: ±1

Small ℓ^1 penalty

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

Big-M constraint: ℓ^0 norm

If we know that $0 \leqslant x \leqslant M$, then we can use $z = \mathbf{1}_{x>0}$ in an ILP: it is the smallest $z \in \{0, 1\}$ such that $x \leqslant Mz$.

We can then minimize $\|\beta\|_0$, if we know that $\forall i \ \ |\beta_i| < M$:

$$\|\beta\|_0 = \sum z_i$$
$$z \in \{0, 1\}^k$$
$$-Mz \leqslant \beta \leqslant Mz.$$

Note that:

– This is a strict inequality, $x > 0$, not $x \geqslant 0$;

– z should be minimized (it should be a term in the objective function)

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

Big-M constraint: 0-1 loss

If we know that $x \in [\![-M, M]\!]$, we can use $z = \mathbf{1}_{x \geqslant 0}$ in an optimization problem by noticing that, since x is an integer, $z = \mathbf{1}_{x+\gamma>0}$ for an arbitrary $\gamma \in (0, 1)$ and, as before, adding constraints $z \in \{0, 1\}$, $x + \gamma \leqslant z(M + \gamma)$, and minimizing z.

If $y \in \{\pm 1\}$, $\hat{y} \in \mathbf{Z}$, the 0-1 loss is $\text{loss}\big(\text{sign}(\hat{y}), y\big) = \mathbf{1}_{y\hat{y} \leqslant 0}$. To minimize it, if we know that $|y\hat{y}| \leqslant M$, miminize z, where $z \in \{0, 1\}$ and $\gamma - y\hat{y} \leqslant z(\gamma + M)$.

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

SLIM

$$
\begin{aligned}
&\text{Find} && \beta \in [\![-B, B]\!]^k \\
& && z \in \{0,1\}^k \\
& && \ell \in \{0,1\}^n \\
&\text{To minimize} && \textstyle\sum_i \ell_i + \lambda \sum_j z_j + \varepsilon \left\|\beta\right\|_1 \\
&\text{Such that} && -Bz \leqslant \beta \leqslant Bz \\
& && \gamma - y \odot (X\beta) \leqslant (\gamma + kB)\ell \\
&\text{Where} && \ell_i = \text{loss}(y_i, \beta' x_i) \\
& && \textstyle\sum z_j = \left\|\beta\right\|_0
\end{aligned}
$$

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

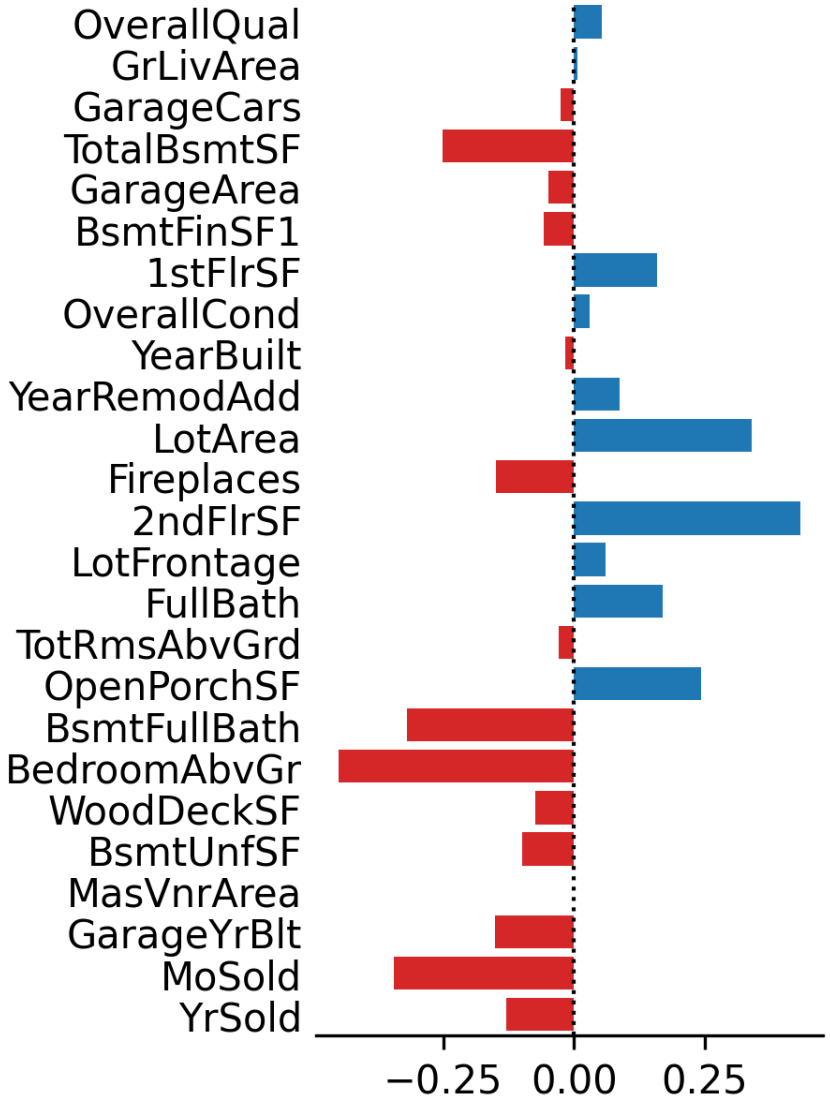
```
# Untested -- do not use

B = 5        # Maximum value of the coefficients
C0 = .001    # L⁰ penalty -- you will need to fine-tune this
C1 = C0/100  # L¹ penalty -- we want the L¹ penalty to be smaller than the L⁰ penalty: it is only there to avoid ties
gamma = .5

assert np.all( np.isin( Xb, [-1,+1] ) ), "Data should be binary, ±1"
assert np.all( np.isin( yb, [-1,+1] ) ), "Data should be binary, ±1"
n,k = Xb.shape

beta = cp.Variable(k, integer=True)
z    = cp.Variable(k, boolean=True)
L    = cp.Variable(n, boolean=True)
loss = cp.mean(L)
penalty0 = cp.sum(z)
penalty1 = cp.norm1(beta)
objective = loss + C0 * penalty0 + C1 * penalty1
constraints = [
    -B * z <= beta, beta <= B * z,
    gamma + cp.multiply( - yb , Xb @ beta ) <= (gamma+k*B) * L
]
prob = cp.Problem( cp.Minimize( objective ), constraints )
result = prob.solve(
    solver = cp.MOSEK,
    mosek_params = {
        'MSK_DPAR_OPTIMIZER_MAX_TIME':  5.0,  # In seconds; you should be more patient
    },
)

print( f"Status: {prob.status}" )
print( f"Time:   {int( prob.solver_stats.solve_time )} seconds" )
print( f"loss       = {cp.sum(L).value.astype(int)} / {len(yb)} = {loss.value:.3f}" )
print( f"L0 penalty = {np.sum(z.value).astype(int)}" )
print( f"objective  = {objective.value:.3f}" )
print( f"beta = {beta.value.astype(int)}" )
```

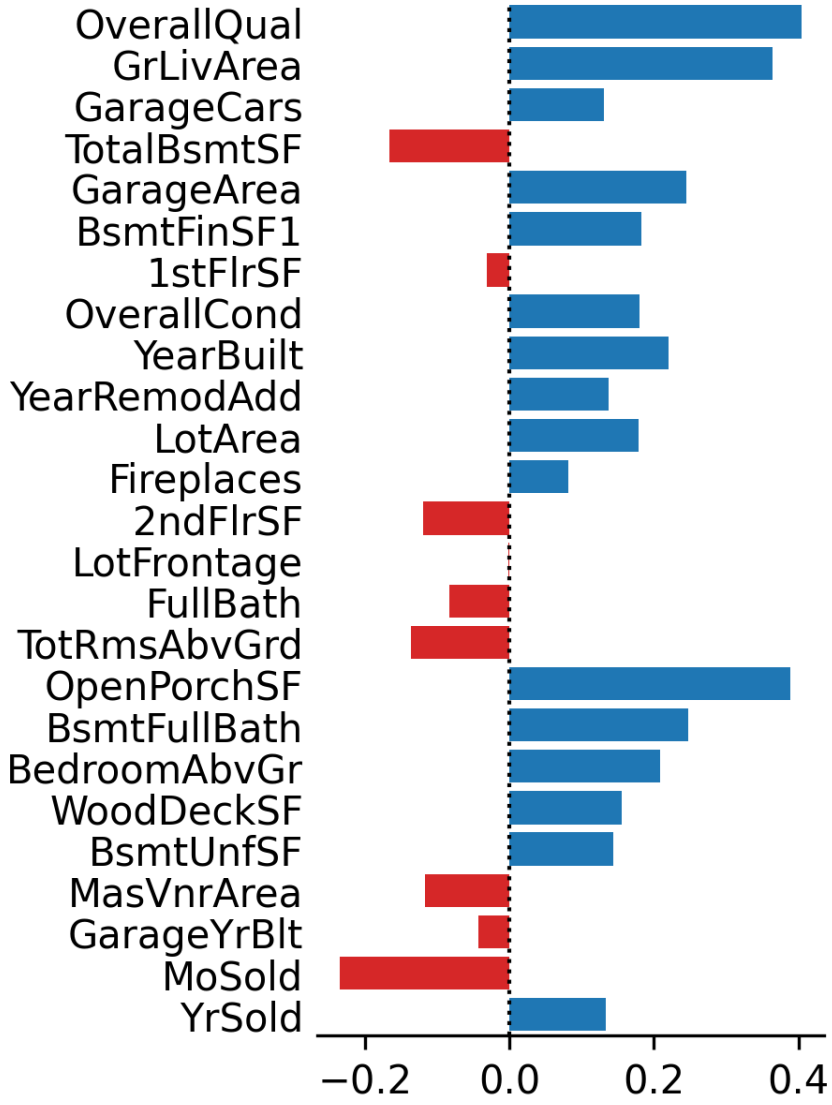
Gradient boosting libraries

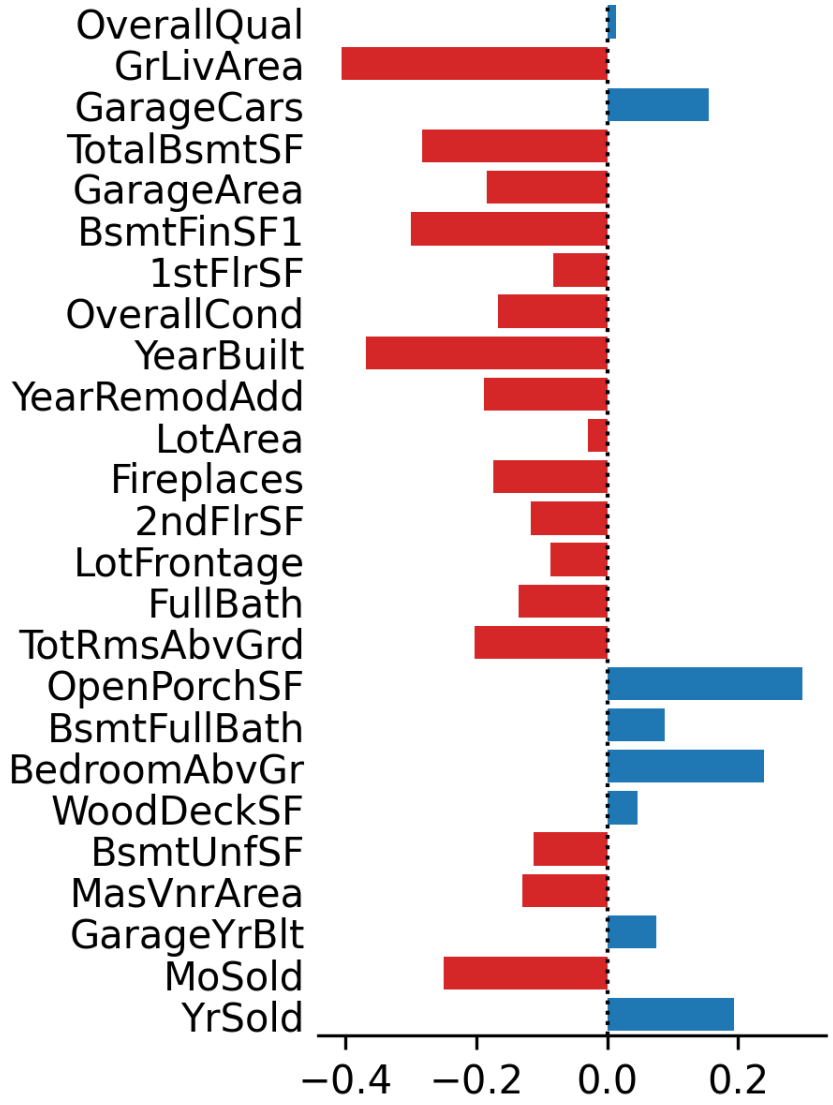
If in doubt, prefer catboost:

- **XGBoost**: uses stumps
- **Lightgbm**: faster, more scalable
- **Catboost**: better handling of categorical values, less prone to overfitting

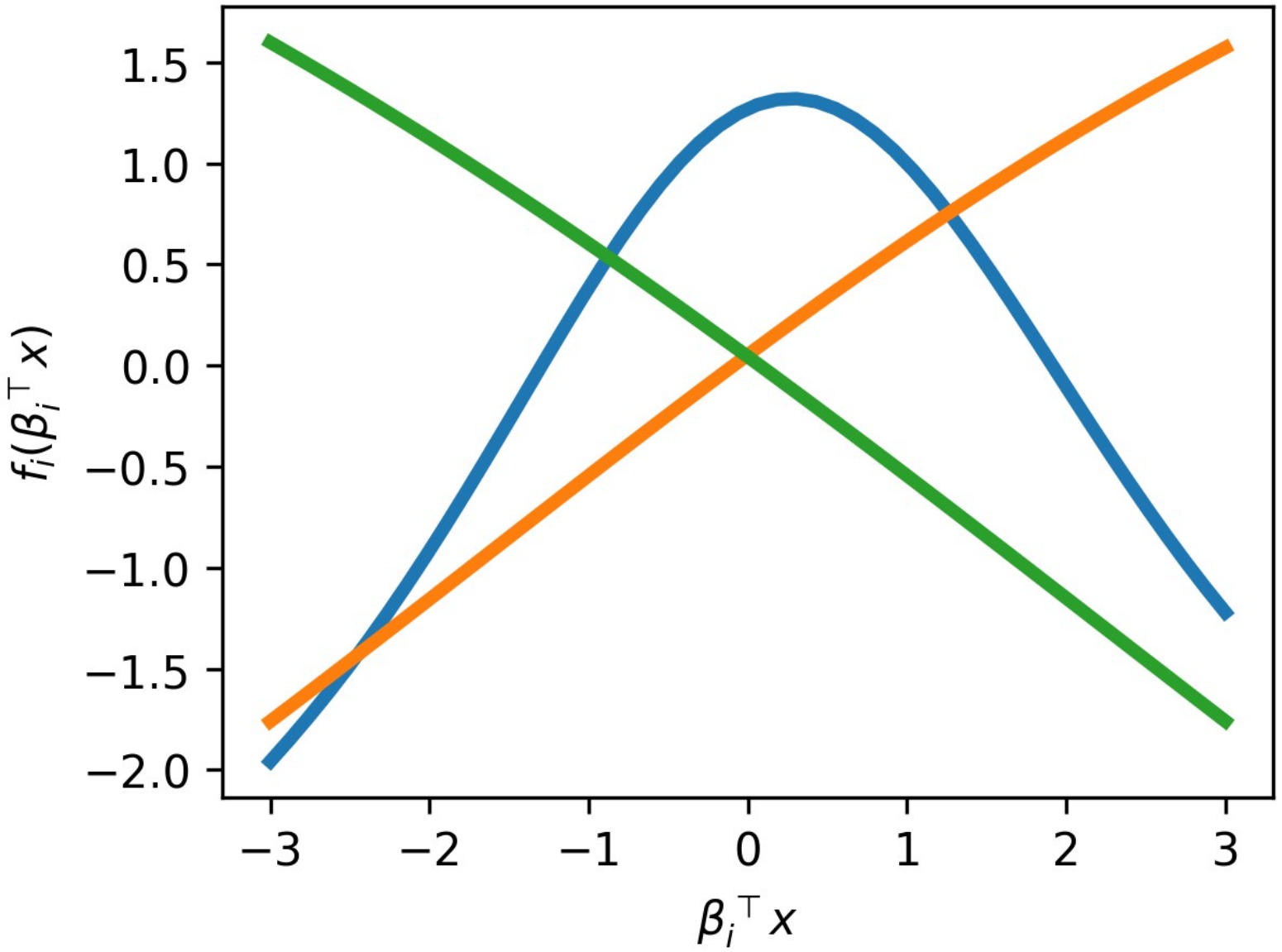
Additive Index Model (AIM)



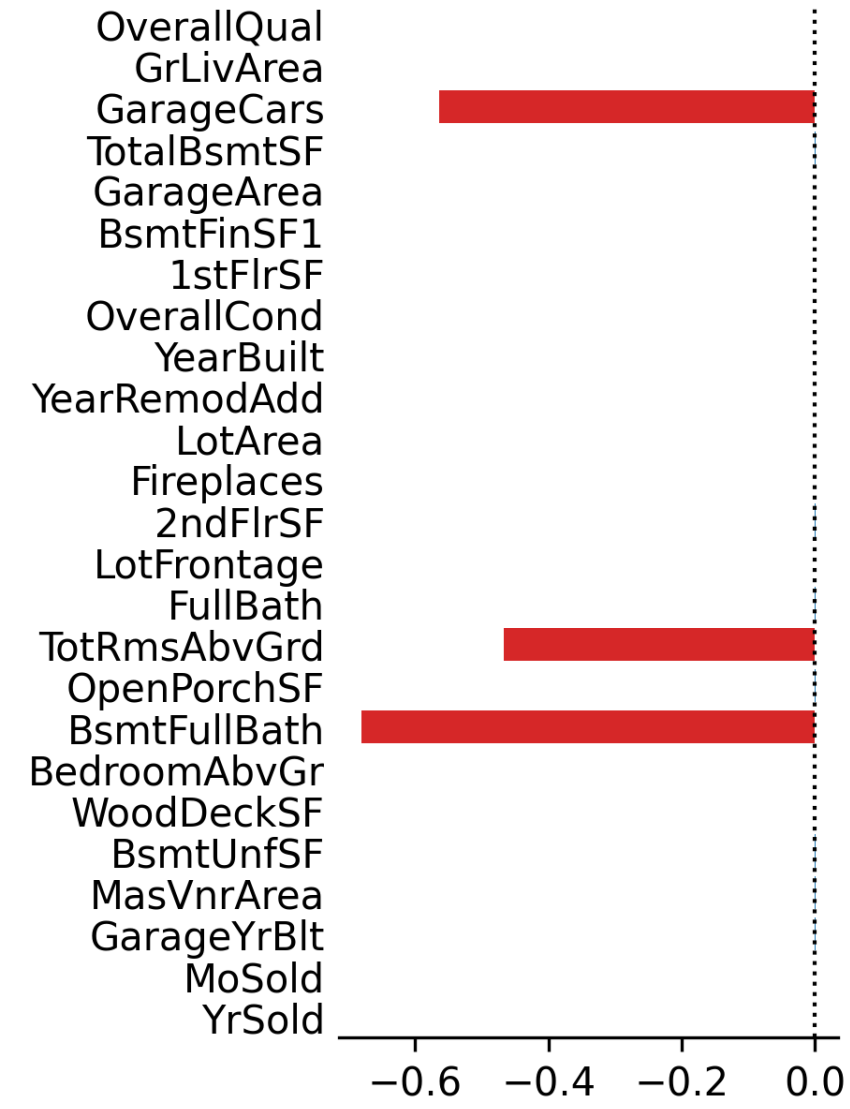


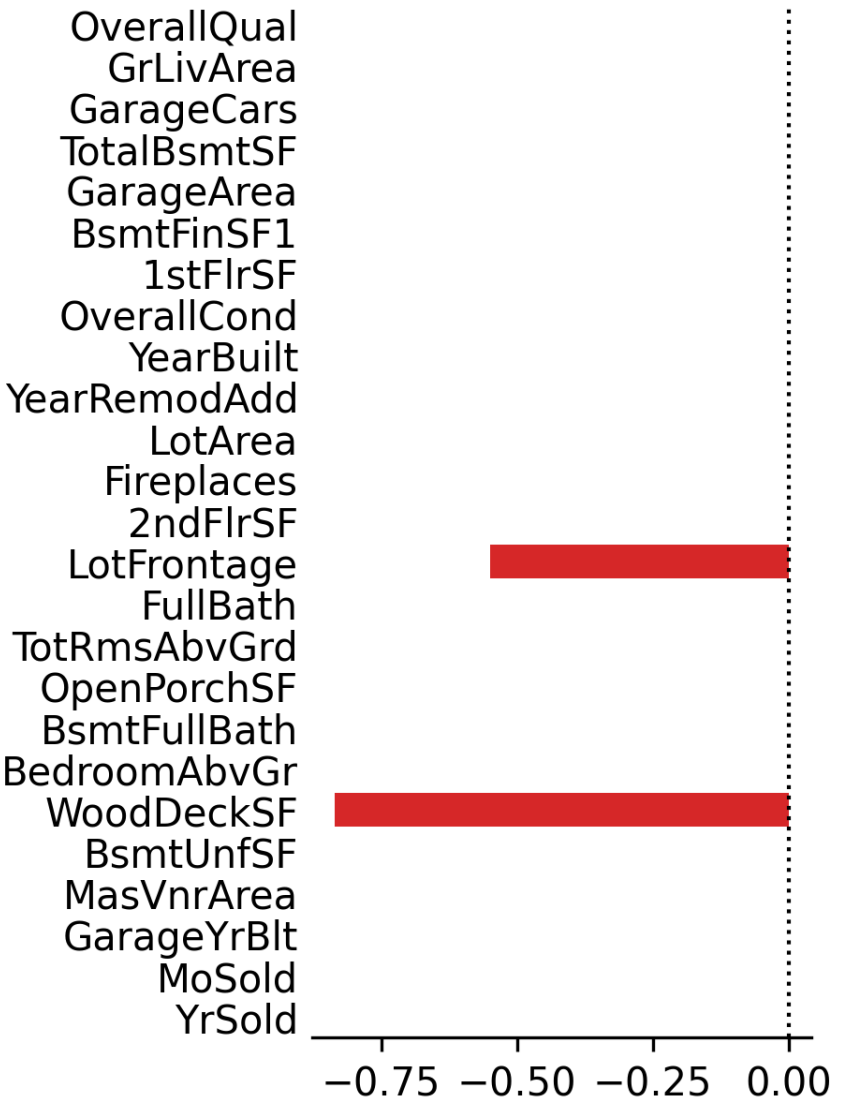


Additive Index Model (AIM)

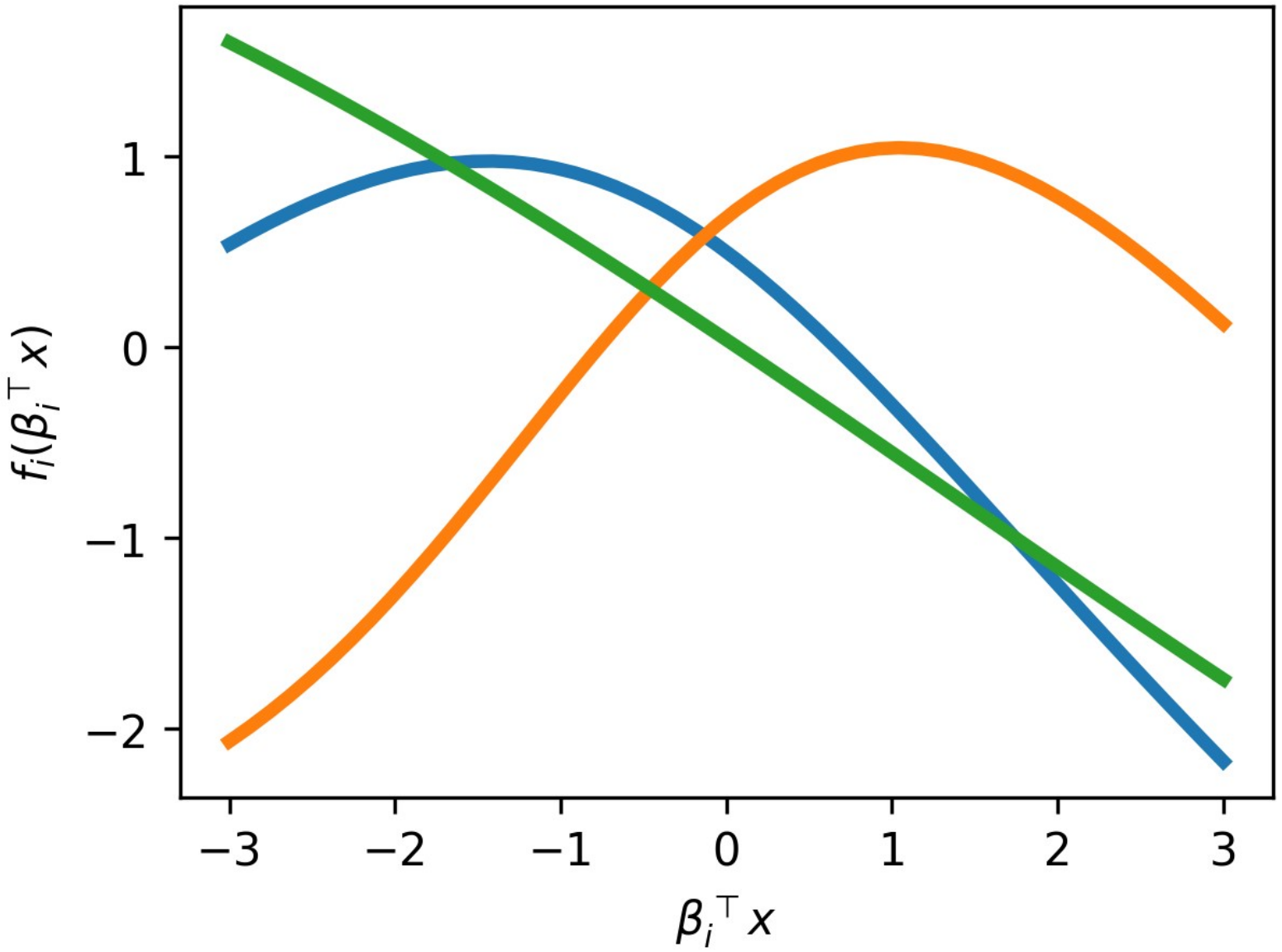


Additive Index Model (AIM)





Additive Index Model (AIM)



Proximal operators

$$\operatorname{prox}_f(v) = \operatorname*{Argmin}_x f(x) + \tfrac{1}{2}\|x - v\|_2^2$$

$$\operatorname{prox}_{\lambda\|\cdot\|_1}(v) = \operatorname{sign}(x)\big(|x| - \lambda\big)$$

A sparse-group lasso (N. Simon et al., 2013)

Proximal gradient descent

Gradient descent, to minimize f:

$$\begin{aligned} x_{\text{new}} &= \operatorname*{Argmin}_z f(x) + \nabla f(x)^\top (z-x) + \tfrac{1}{2}(z-x)^\top \nabla^2 f(x)(z-x) \\ &\approx \operatorname*{Argmin}_z f(x) + \nabla f(x)^\top (z-x) + \frac{1}{2t}\|z-x\|^2 \text{ if } \nabla^2 f \approx \frac{1}{t} I \\ &= x - t\nabla f(x) \end{aligned}$$

Proximal gradient descent, to minimize $f+g$:

$$\begin{aligned} x_{\text{new}} &= \operatorname*{Argmin}_z f(x) + \nabla f(x)^\top (z-x) + \frac{1}{2t}\|z-x\|^2 + g(z) \\ &= \operatorname*{Argmin}_z \frac{1}{2t}\left\|z - \big(x - t\nabla f(x)\big)\right\|_2^2 + g(x) \\ &= \operatorname{prox}_{tg}\big(x - t\nabla f(x)\big) \end{aligned}$$

A sparse-group lasso (N. Simon et al., 2013)

Varying coefficient models

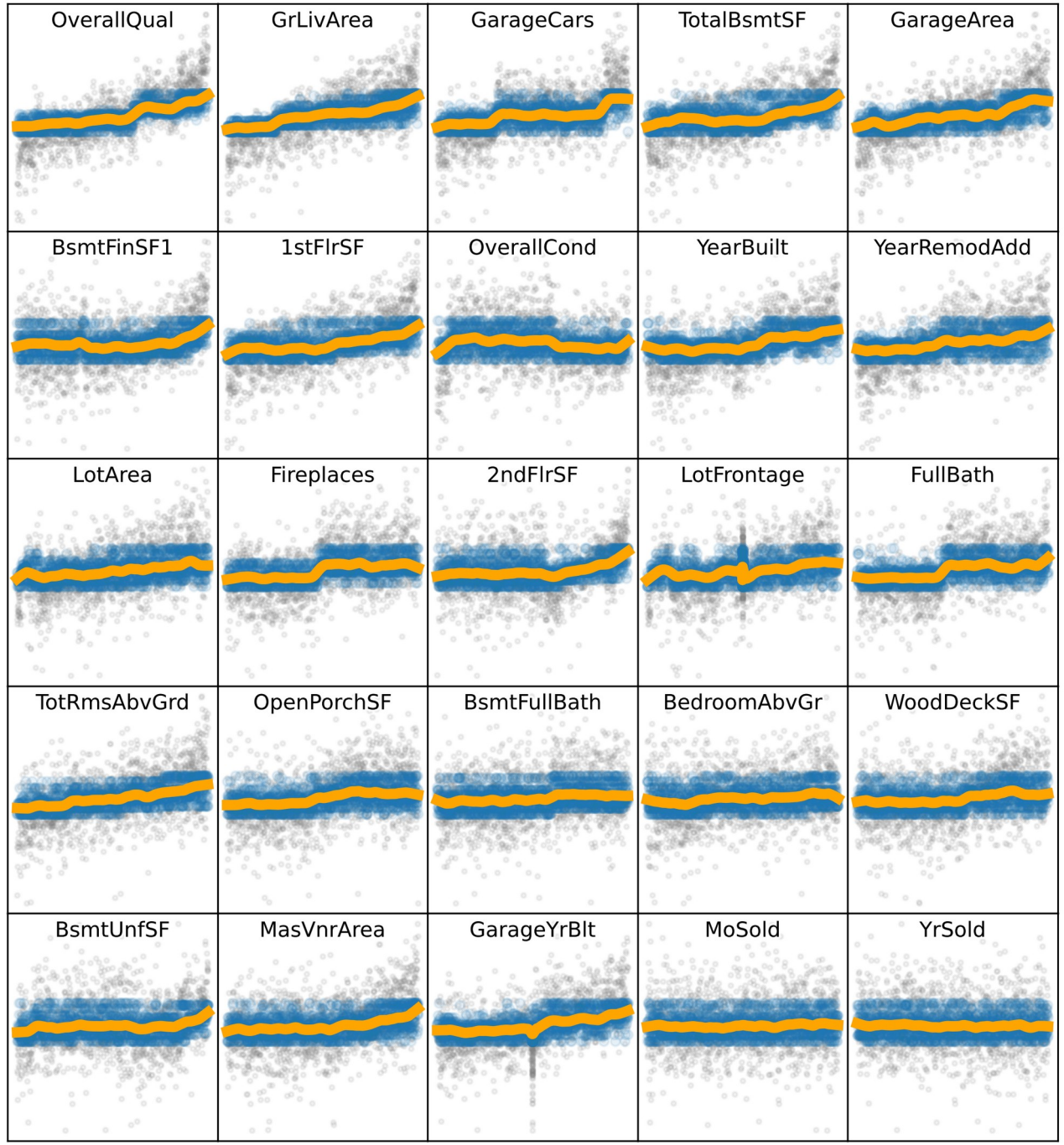
I tried to add a penalty on

$$\left\| \frac{\partial \theta}{\partial x} \right\|$$

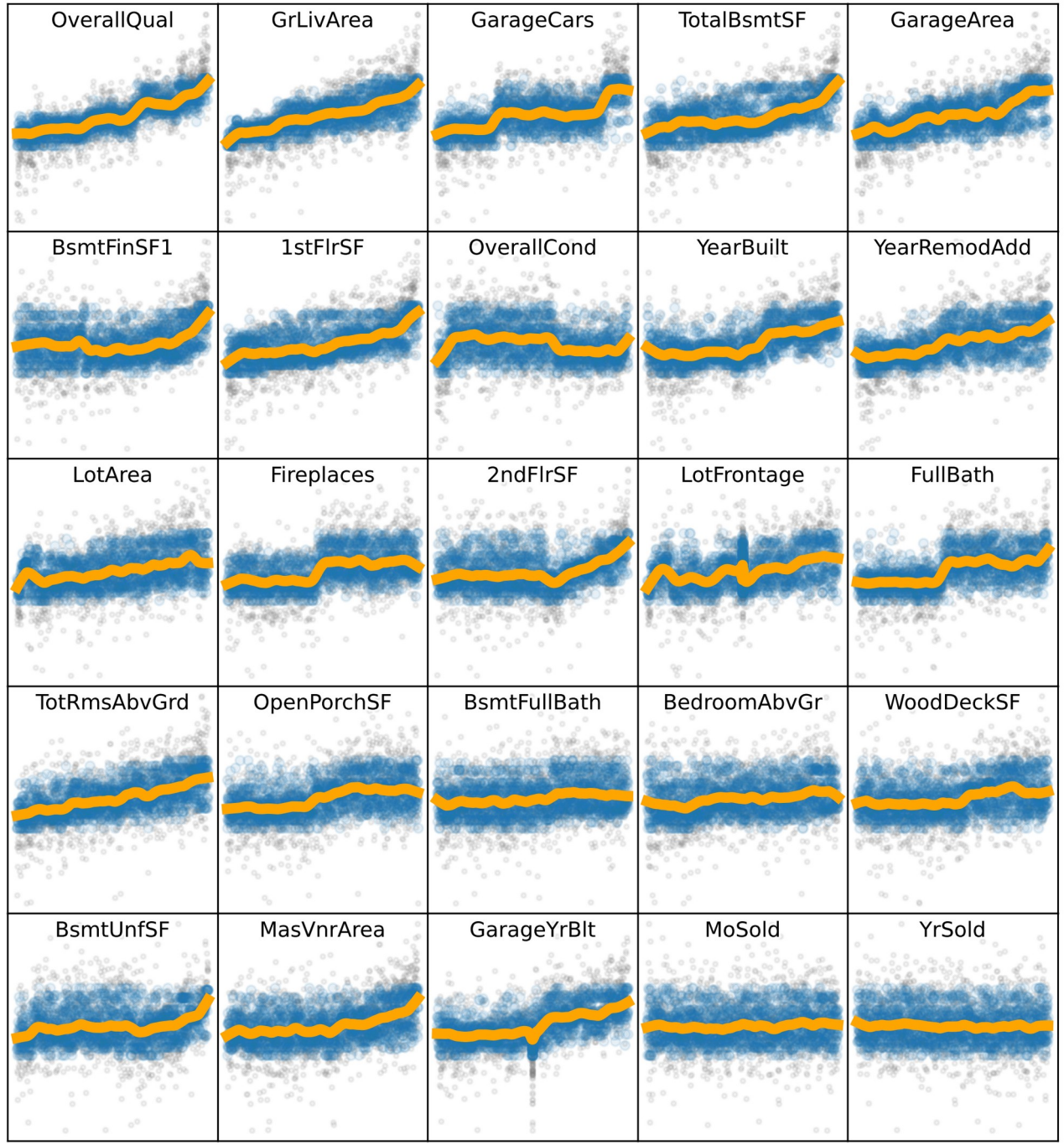
- or $\sum_{i \sim j} \|\theta(x_i) - \theta(x_j)\|$
- or $\left\| \frac{\partial f}{\partial x} - \theta \right\|$

Regularization paths

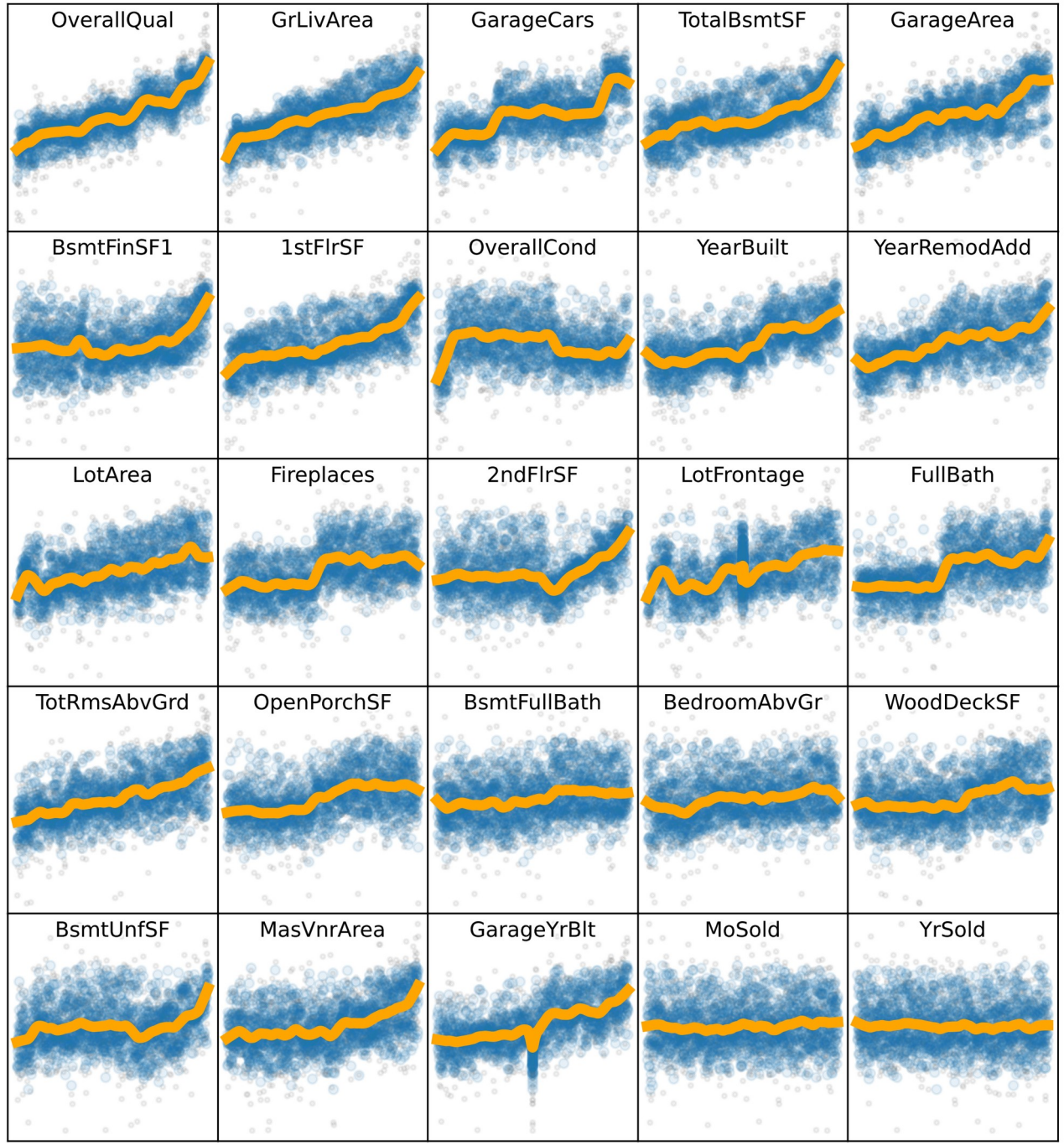
CatBoostRegressor k=1



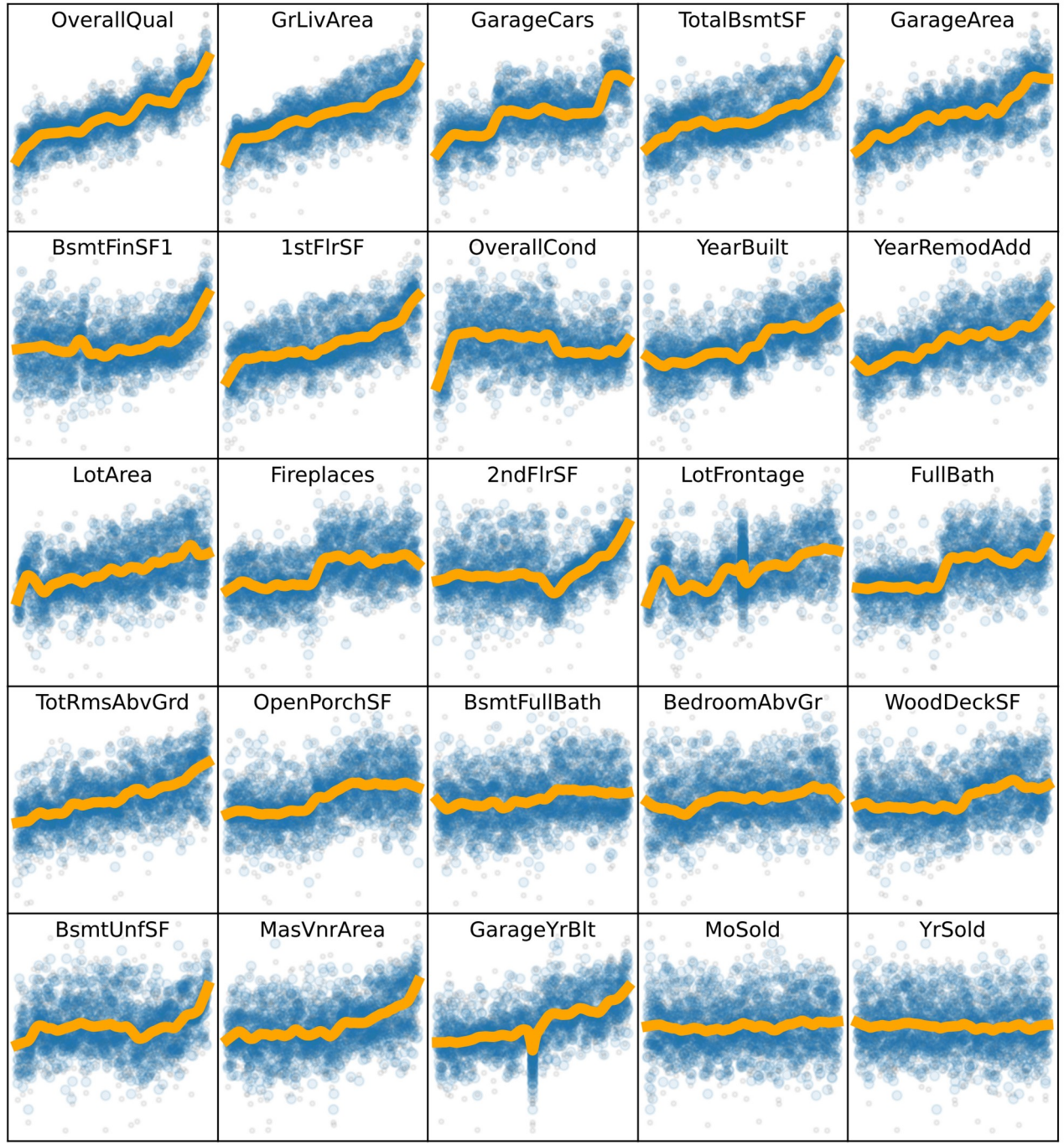
CatBoostRegressor k=2



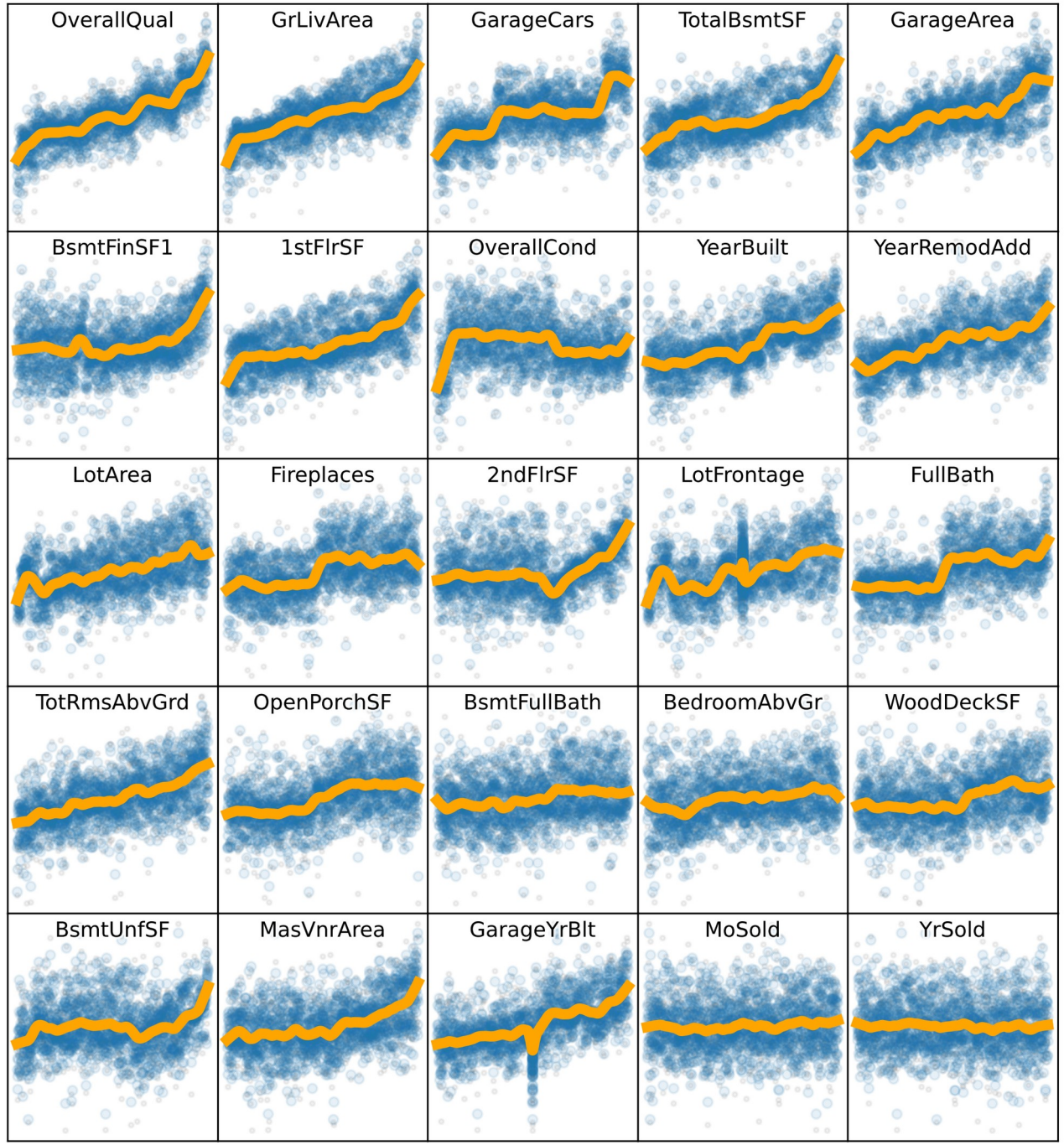
CatBoostRegressor k=5



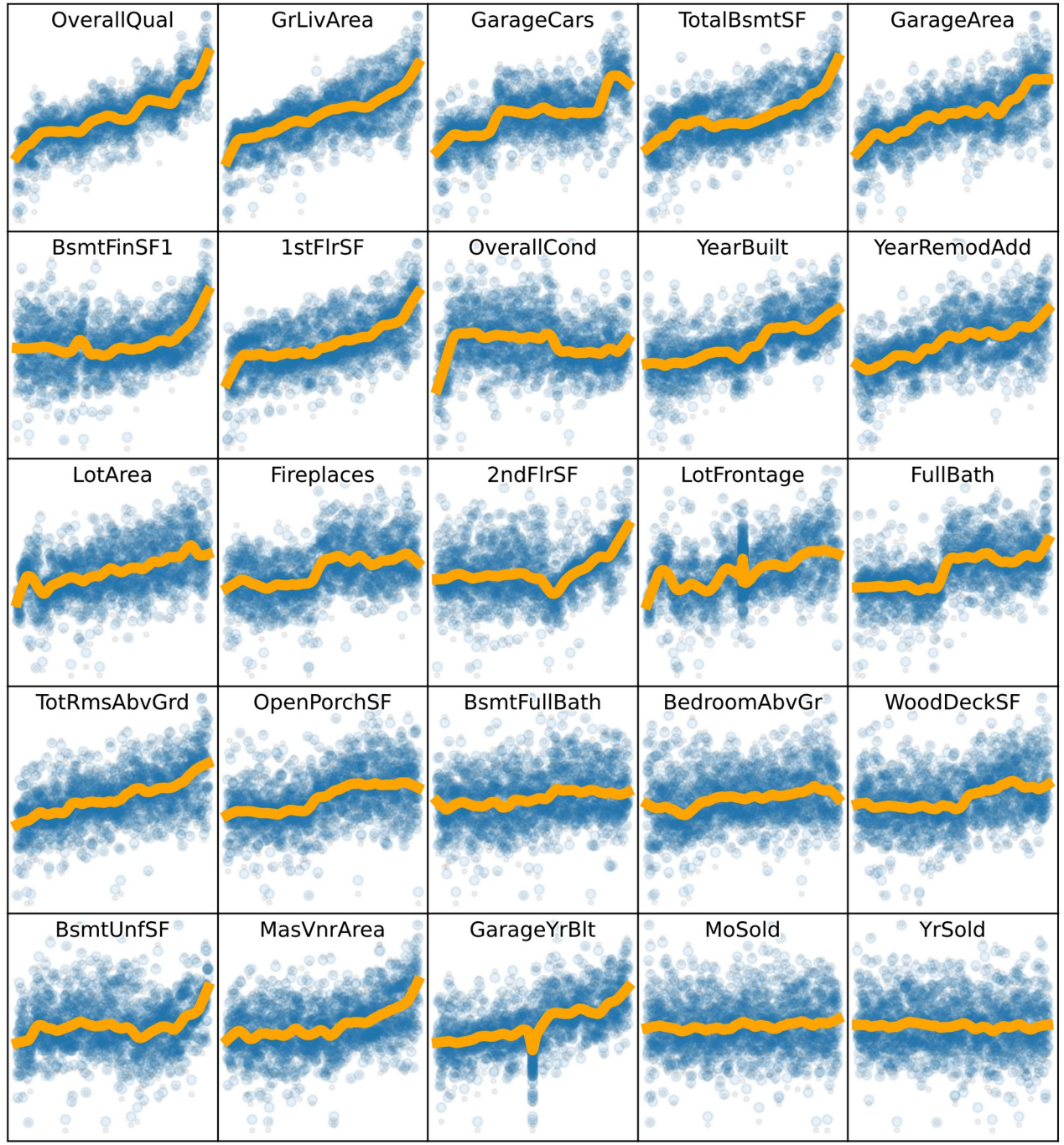
CatBoostRegressor k=10



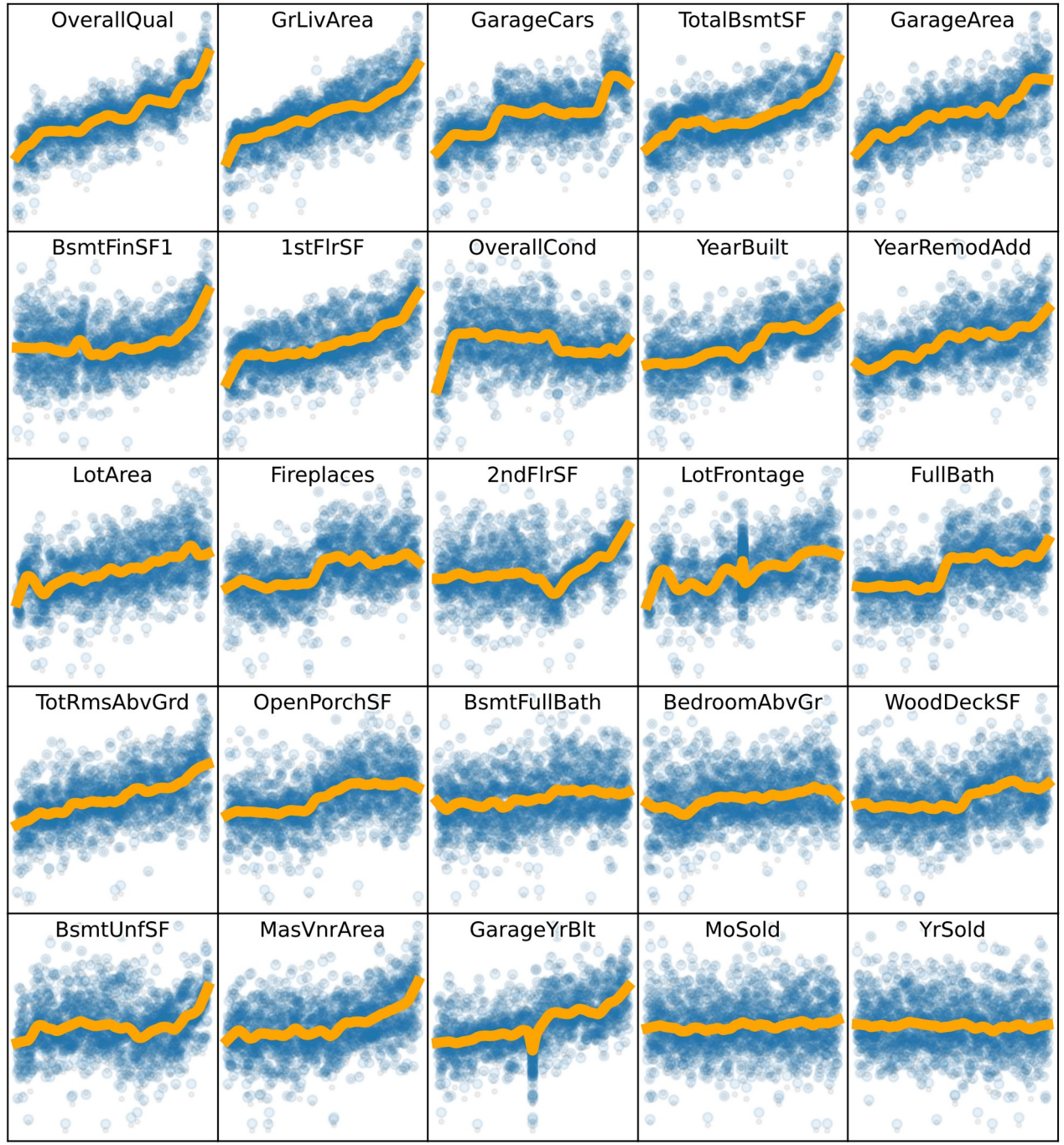
CatBoostRegressor k=20



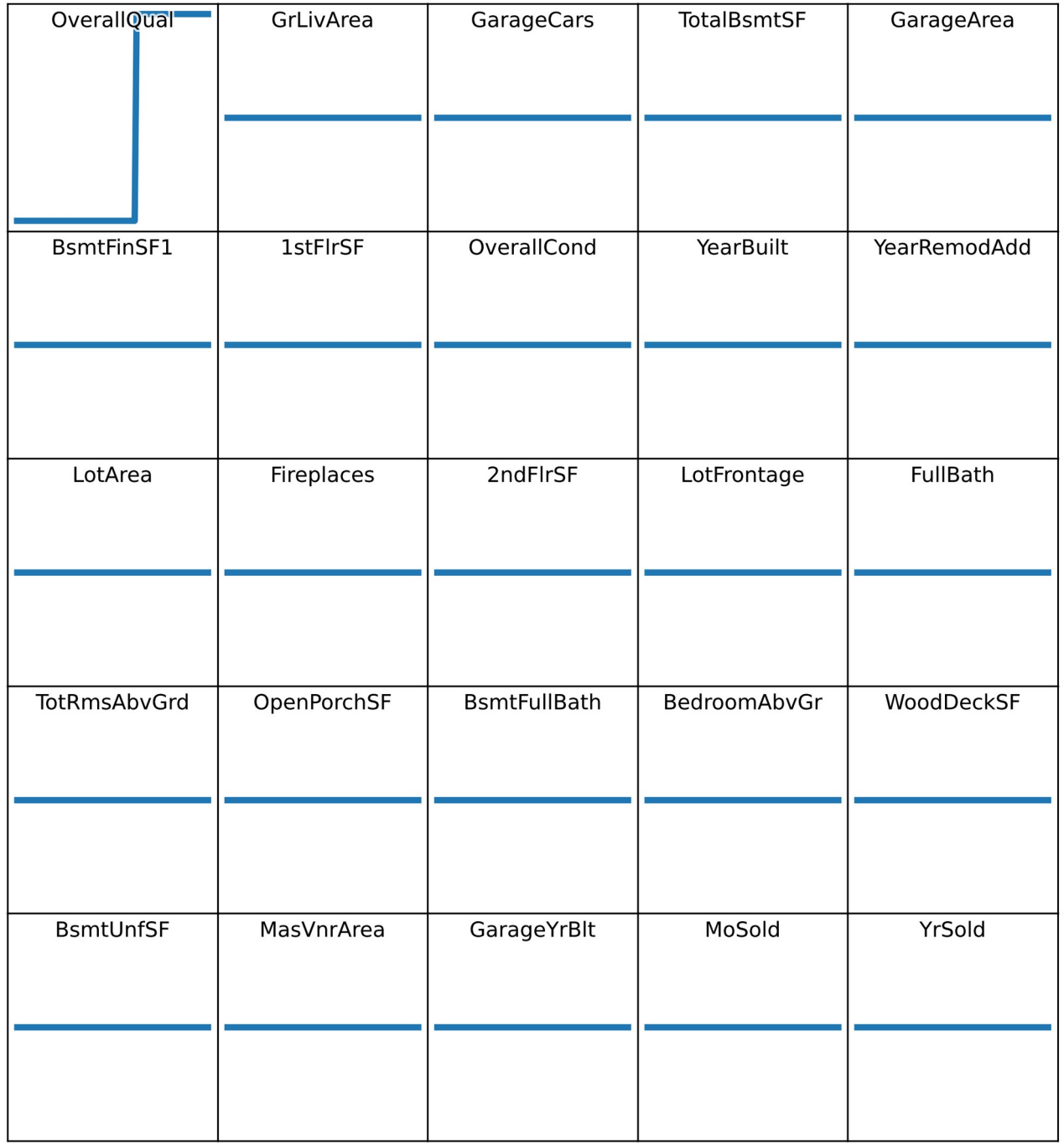
CatBoostRegressor k=50



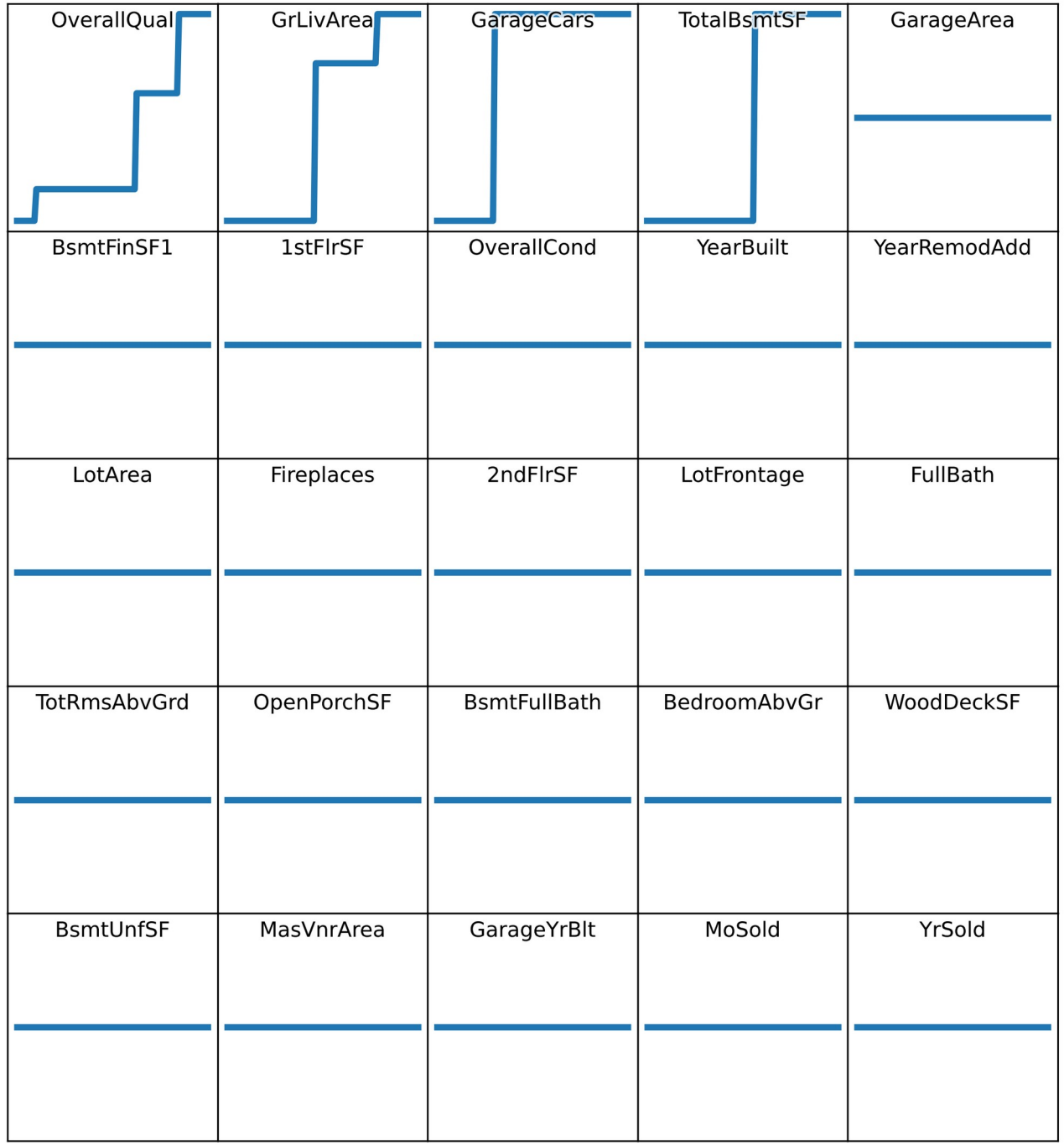
CatBoostRegressor k=100



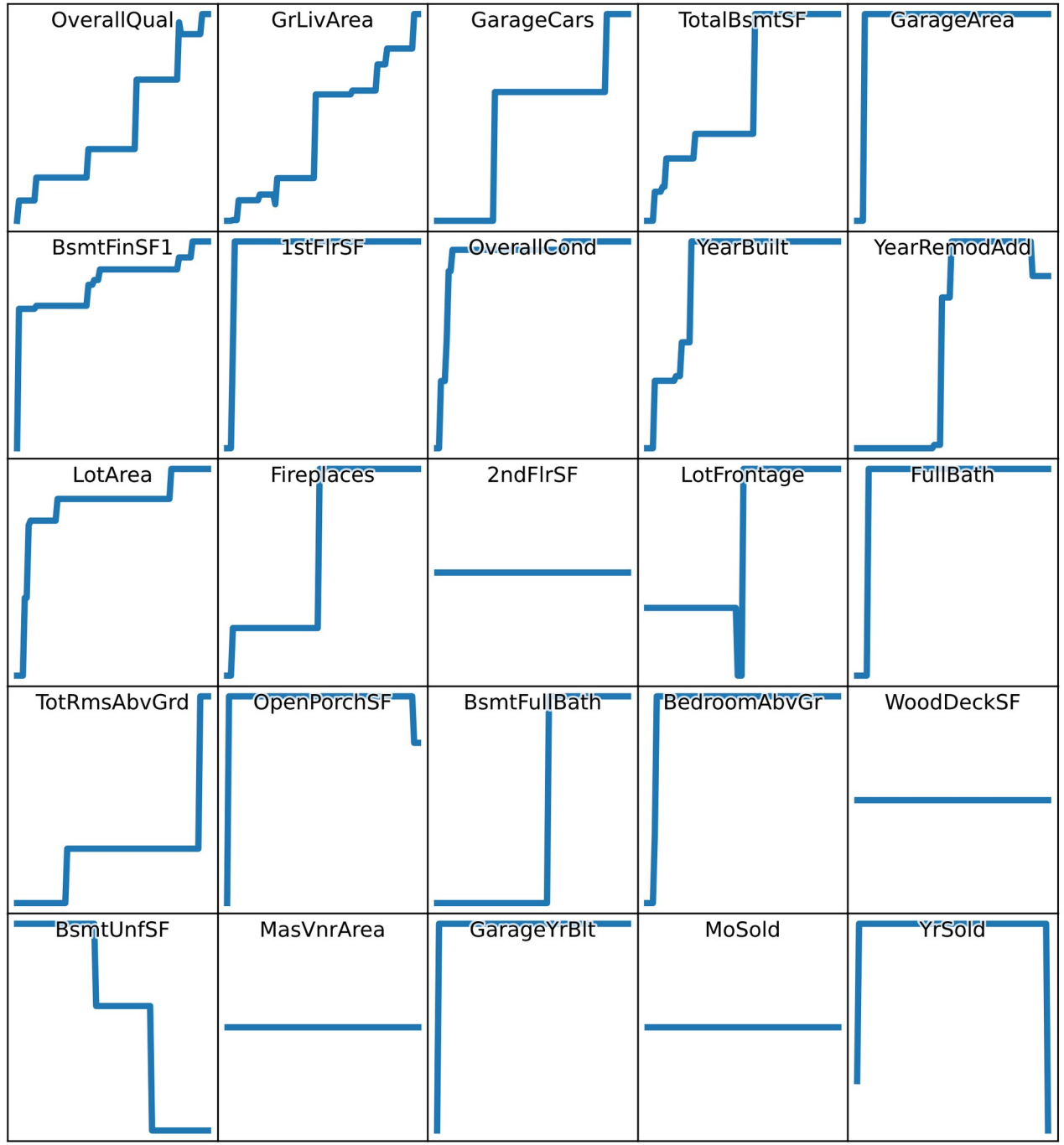
XGBRegressor k=2



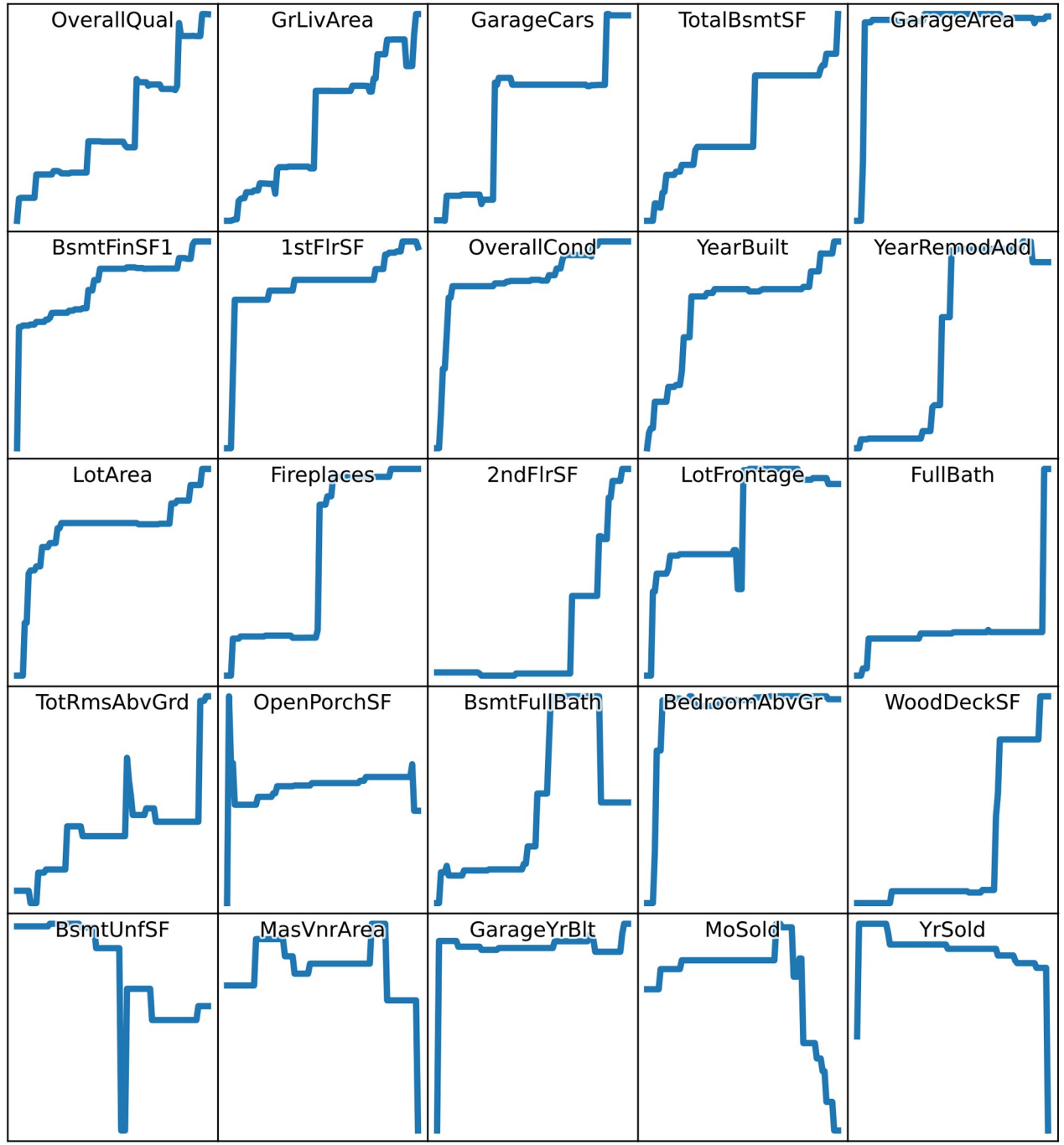
XGBRegressor k=5



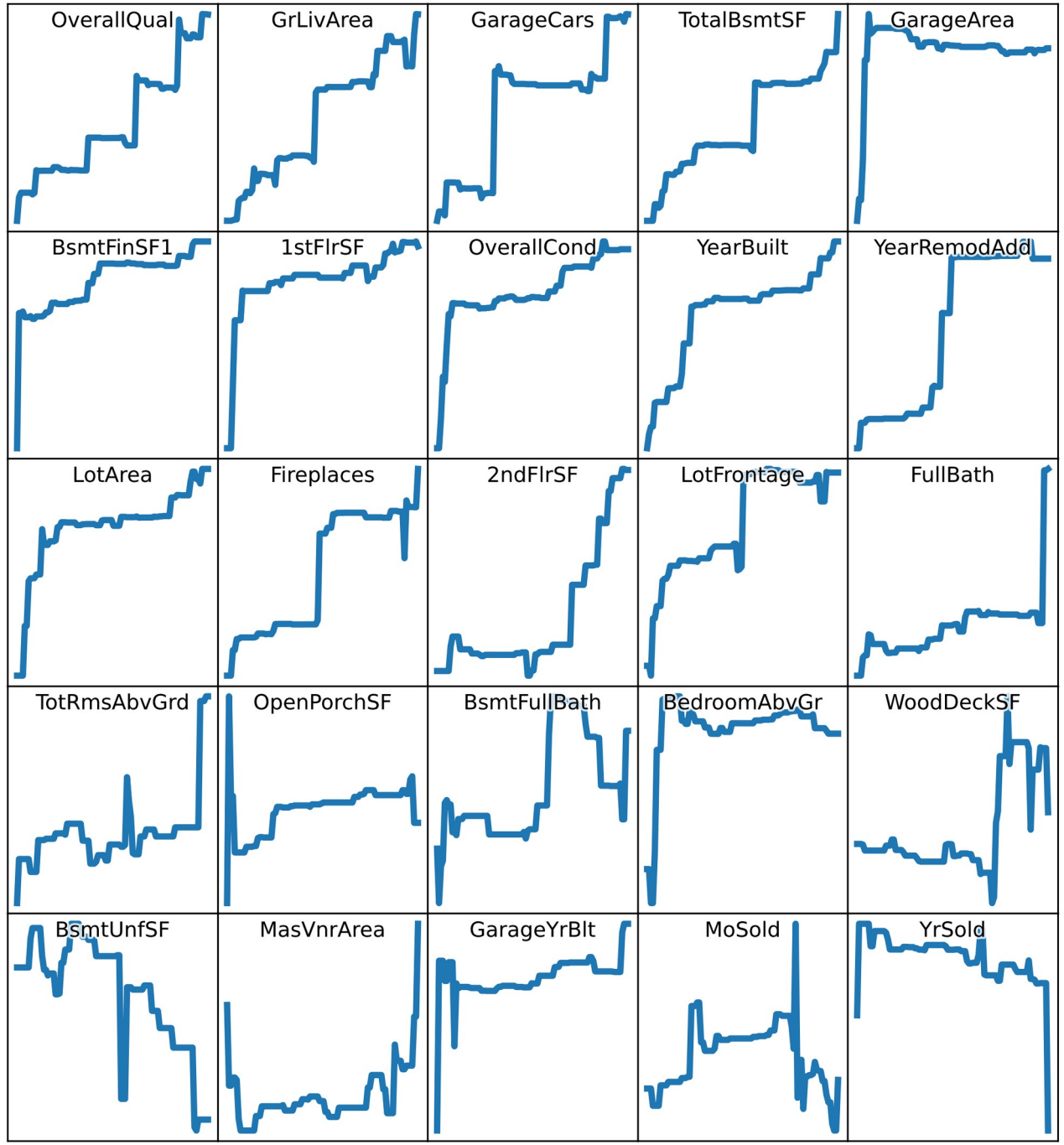
XGBRegressor k=10



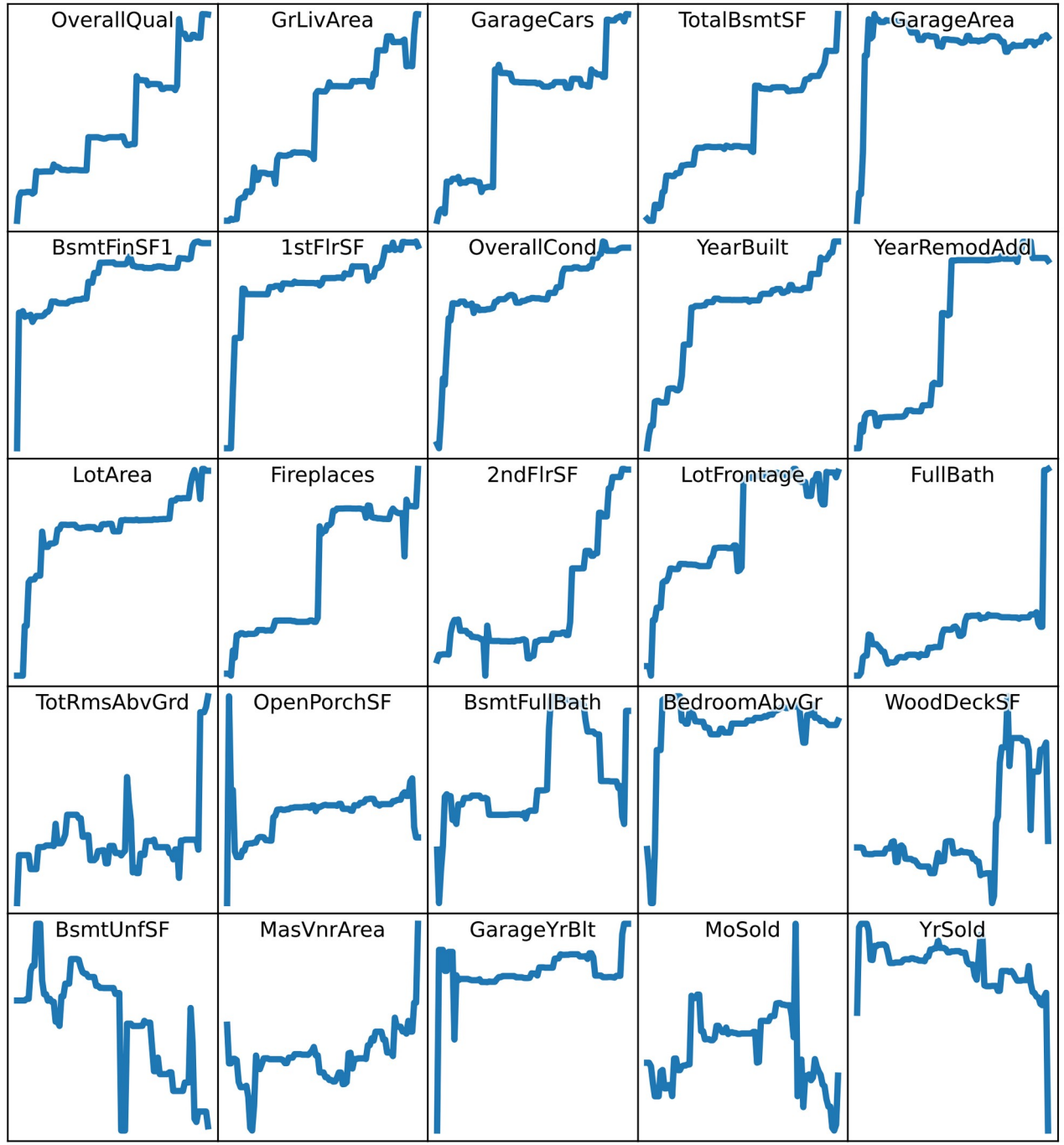
XGBRegressor k=20



XGBRegressor k=50



XGBRegressor k=100



Unused Slides

$$
\begin{aligned}
&\text{Find} && \beta \in [\![-B, B]\!]^k \\
& && z \in \{0,1\}^k \\
& && \ell \in \{0,1\}^n \\
&\text{To minimize} && \textstyle\sum_i \ell_i + \lambda \sum_j z_j + \varepsilon \left\|\beta\right\|_1 \\
&\text{Such that} && -Bz \leqslant \beta \leqslant Bz \\
& && \gamma - y \odot (X\beta) \leqslant (\gamma + kB)\ell \\
&\text{Where} && \ell_i = \text{loss}(y_i, \beta' x_i) \\
& && \textstyle\sum z_j = \left\|\beta\right\|_0
\end{aligned}
$$

Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

Definitions

- No definition/measure of “interpretability” or “explainability”
- Some authors use those terms interchangeably
- Interpretable model = interpretable by construction
- Explanation methods = post hoc, works on trained, blackbox models

Interpretability

- Additivity
- Sparsity
- Linearity
- Smoothness
- Monotonicity
- Visualizability
- Projection
- Segmentation

Additive Models

- GAM
- AIM
- GA2M (GAM with interactions)
- GAM Boosting

Before: Interpretable AI

- (Monotonic) falling rule lists, decision trees
- Sparse linear models, SLIM, sparse input neural net, fused lasso, graph sparsity
- GAM, GA2M, neural GAM, GAM boosting
- Varying coefficient models, self-explaining neural nets
- Attentive mixture of experts
- Sign and monotonicity constraints
- Symbolic regression
- Disentangled representations

Before: Interpretable AI

Prior, inductive bias:

- Decision trees, rule lists
- Linear regression, (graph) sparsity
- GAM Boosting
- Linear regression, sign constraints
- Neural networks

Zoo of interpretable models

- Rule-based
- Sparse
- Additive

Code

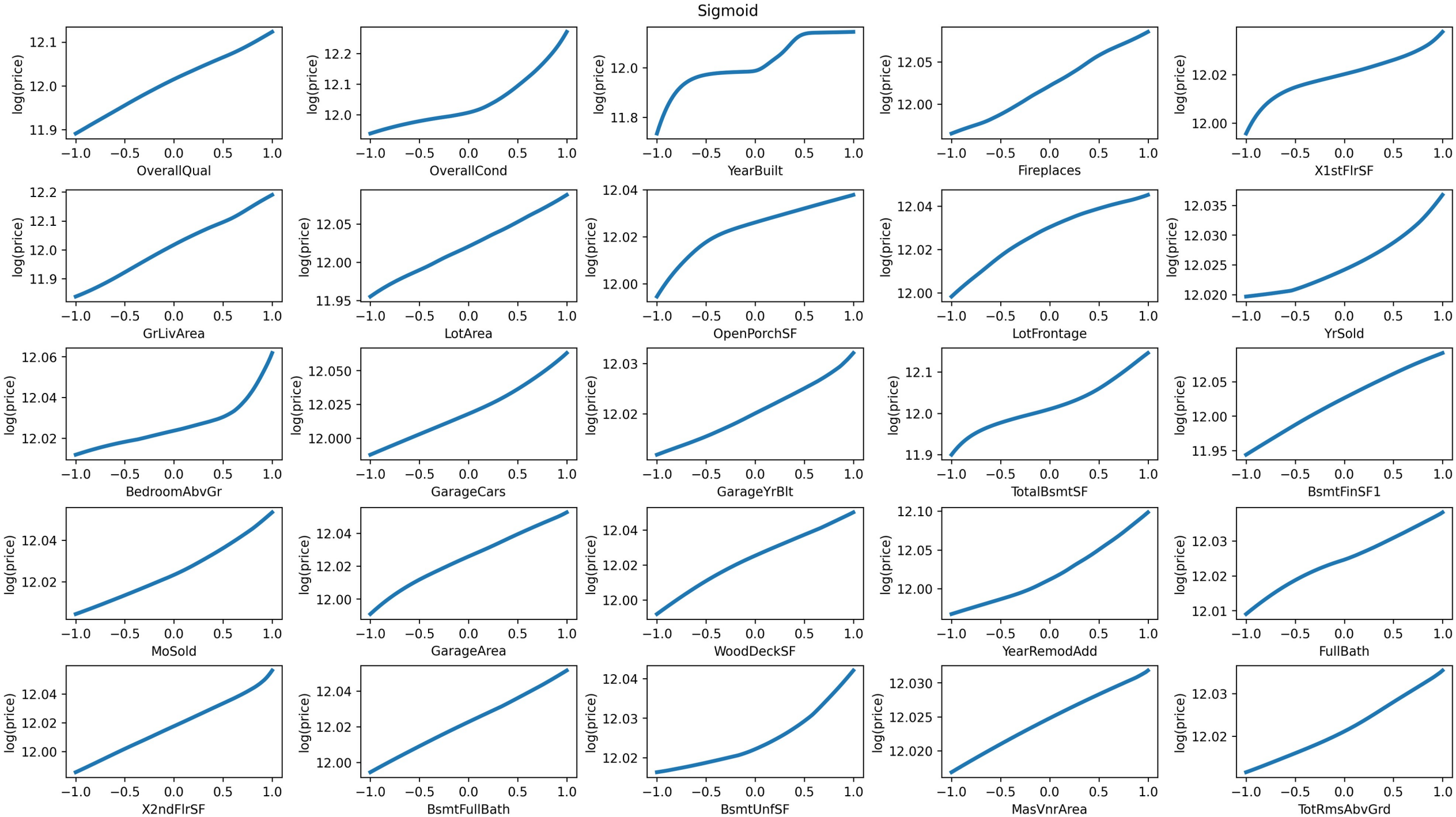
```
# ElasticNet (linear regression with L¹ and L² penalties)
model = sklearn.linear_model.ElasticNet( alpha = alpha )
model.fit( X, y )
yhat = model.predict(X)
```

Code

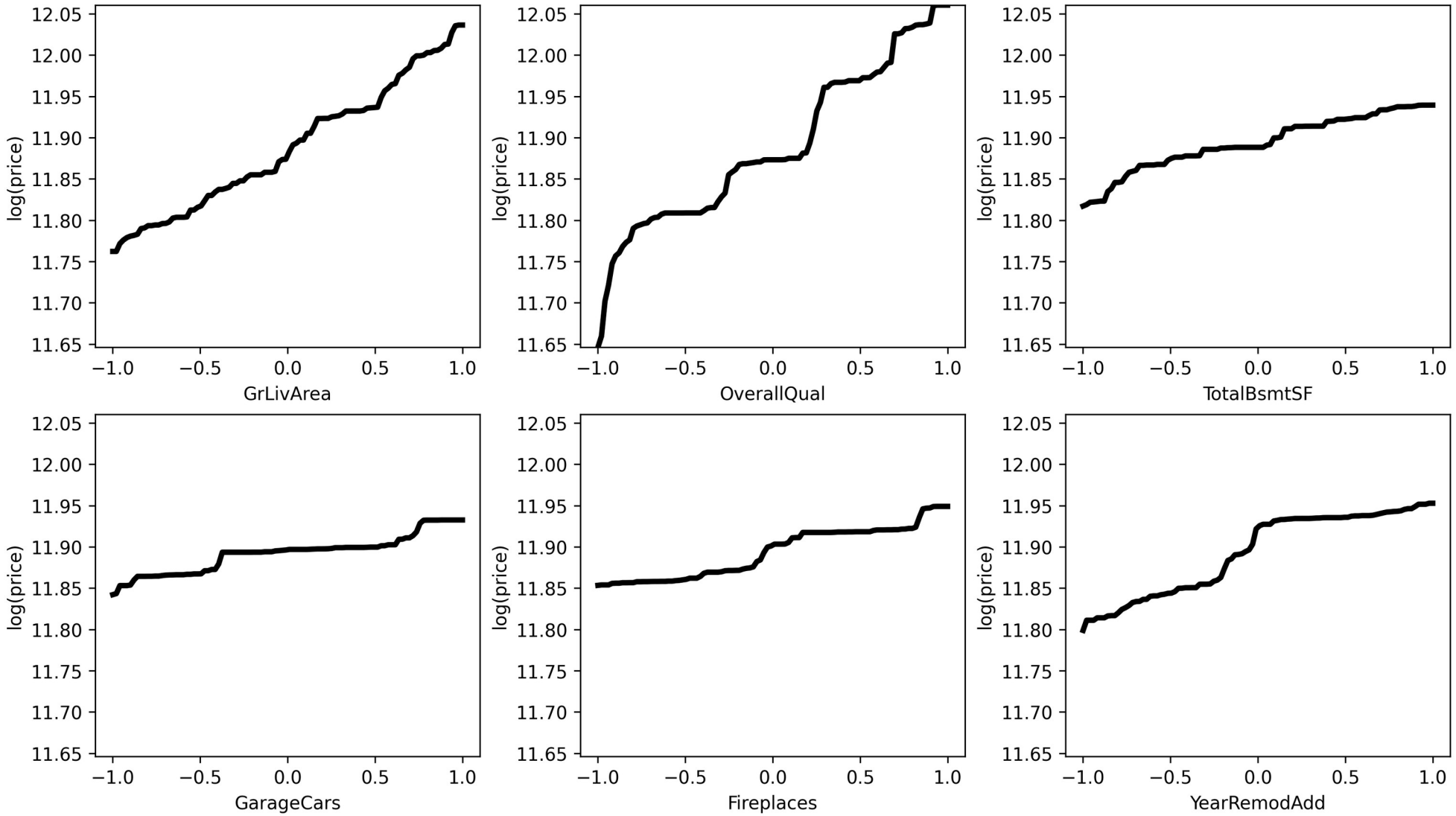
```
error = loss_fn( y_hat, y_ )
penalty_12 = torch.linalg.vector_norm( model.fc1.weight, ord=2, dim=0 ).sum()
penalty_1  = torch.linalg.vector_norm( model.fc1.weight, ord=1, dim=0 ).sum()
penalty_2  = (
    torch.linalg.vector_norm( model.fc2.weight, ord=2, dim=None ) ** 2 +
    torch.linalg.vector_norm( model.fc3.weight, ord=2, dim=None ) ** 2
)
loss_smooth = error + lambda_2 * penalty_2
loss_smooth.backward()
optimizer.step()

soft_threshold_( model.fc1.weight, lr * lambda_1 * (1-alpha) )
with torch.no_grad():
    for i in range( model.fc1.weight.shape[1] ):
        w = model.fc1.weight[:,i]
        n = torch.linalg.vector_norm(w, ord=2)
        w.copy_( F.relu( 1 - lr * lambda_1 * alpha / n ) * w )
```

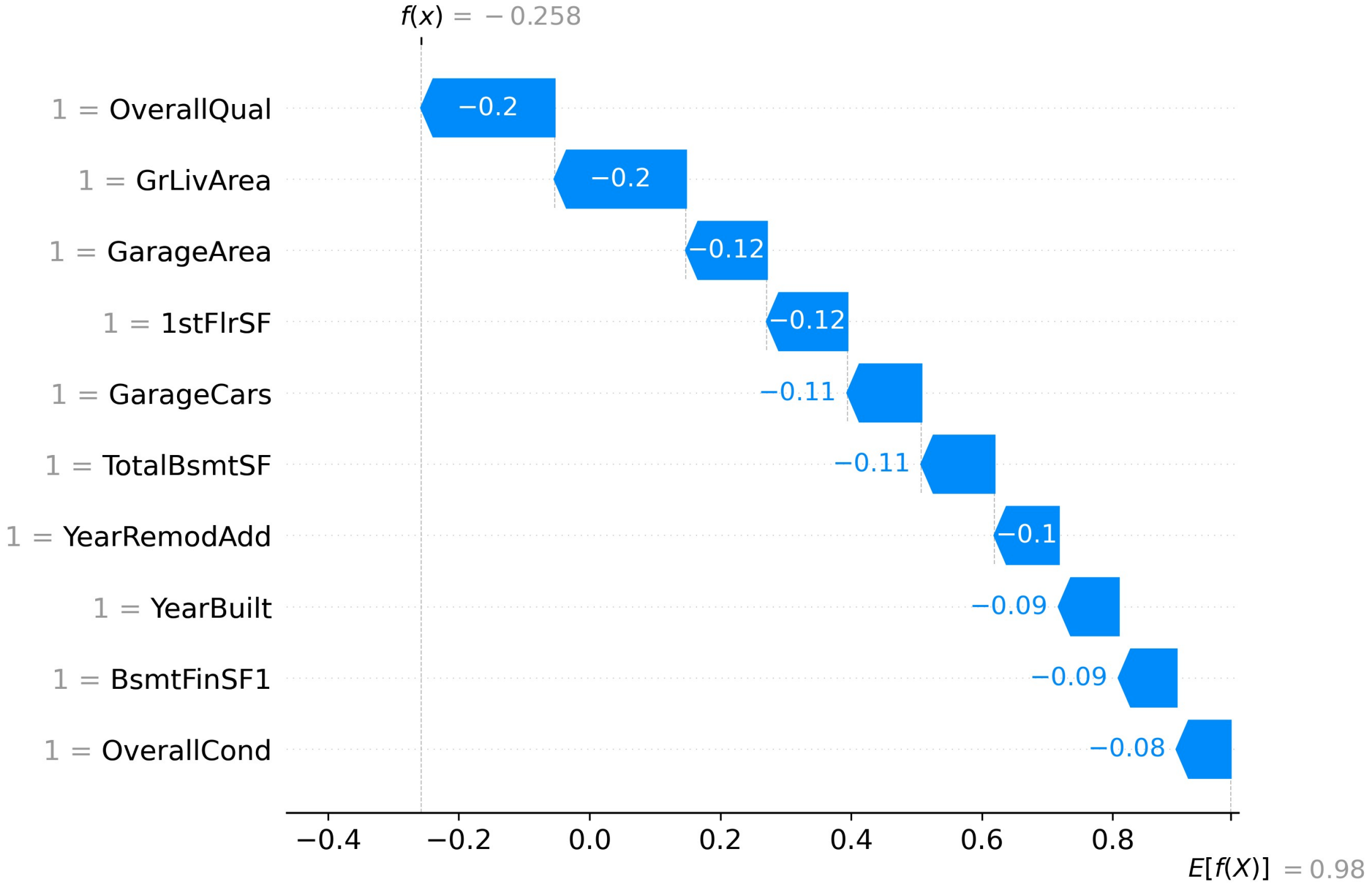
Code

```
class Net(nn.Module):
    def __init__(self, n_input, k1, k2):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(n_input, k1)
        self.fc2 = nn.Linear(k1, k2)
        self.fc3 = nn.Linear(k2, 1)
    def forward(self, x):
        h1 = F.relu( self.fc1(x) )
        h2 = F.relu( self.fc2(h1) )
        return self.fc3(h2)

model = Net( X.shape[1], 200, 200 )
optimizer = optim.Adam( model.parameters() )
loss_fn = nn.MSELoss(reduction='mean')
for epoch in tqdm(range(10_000)):
    optimizer.zero_grad()
    y_hat = model(X_)
    loss = loss_fn( y_hat, y_ )
    loss.backward()
```


Sigmoid
log(price)
OverallQual
OverallCond
YearBuilt
Fireplaces
X1stFlrSF
GrLivArea
LotArea
OpenPorchSF
LotFrontage
YrSold
BedroomAbvGr
GarageCars
GarageYrBlt
TotalBsmtSF
BsmtFinSF1
MoSold
GarageArea
WoodDeckSF
YearRemodAdd
FullBath
X2ndFlrSF
BsmtFullBath
BsmtUnfSF
MasVnrArea
TotRmsAbvGrd

Gradient boosting



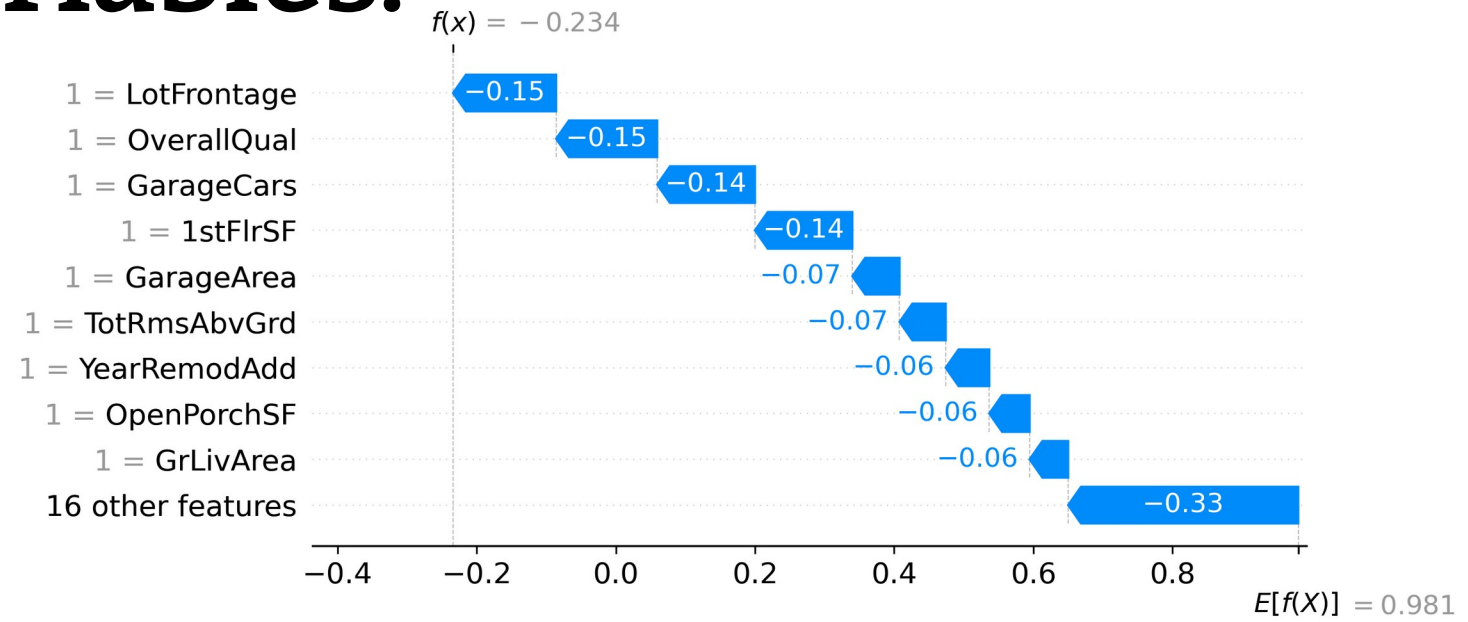

f(x) = −0.258
1 = OverallQual
1 = GrLivArea
1 = GarageArea
1 = 1stFlrSF
1 = GarageCars
1 = TotalBsmtSF
1 = YearRemodAdd
1 = YearBuilt
1 = BsmtFinSF1
1 = OverallCond
−0.2
−0.2
−0.12
−0.12
−0.11
−0.11
−0.1
−0.09
−0.09
−0.08
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
E[f(X)] = 0.98

Shapley Values

- Decomposition of the performance of a model into a sum of contributions of each input variable, obtained by fitting the model on all subsets of variables.

$$A_I = \sum_{i \in I} A_i$$

$$A_i = \underset{S \ni i}{\text{Mean}}\, A_S - A_{S \setminus \{i\}}$$



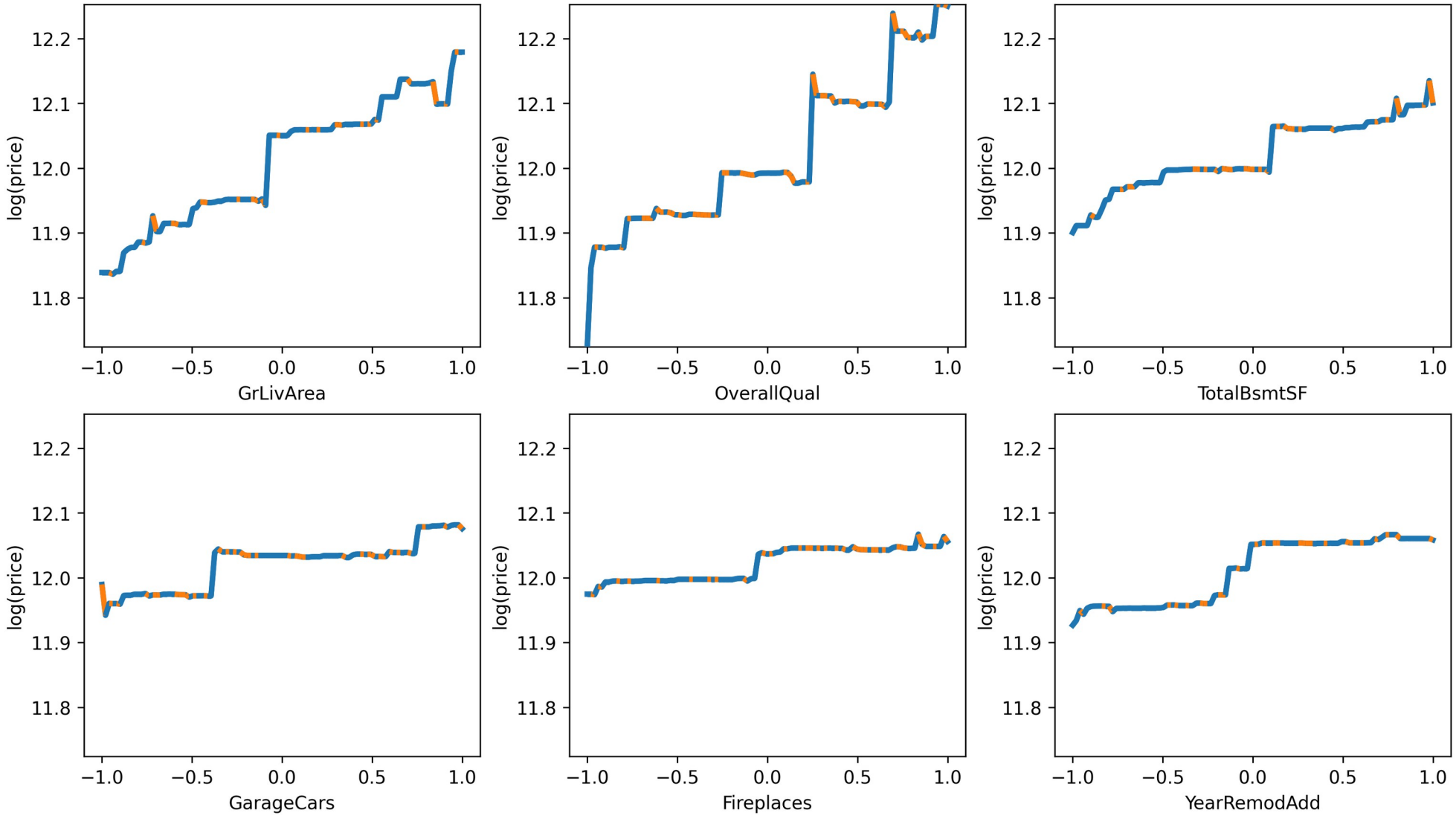
SLIM

- Combine binary inputs with simple integer coefficients

$$
\begin{aligned}
&(\text{uniformity of cell size} \geqslant 3) + \\
&(\text{uniformity of cell shape} \geqslant 3) + \\
&(\text{marginal adhesion} \geqslant 3) + \\
&(\text{single epithelial cell size} \geqslant 3) + \\
&(\text{bare nuclei} \geqslant 3) + \\
&(\text{normal nucleoli} \geqslant 3) + \\
&(\text{bland chromatin} \geqslant 3) + \\
&(\text{mitoses} \geqslant 3) \\
&\geqslant 8
\end{aligned}
$$

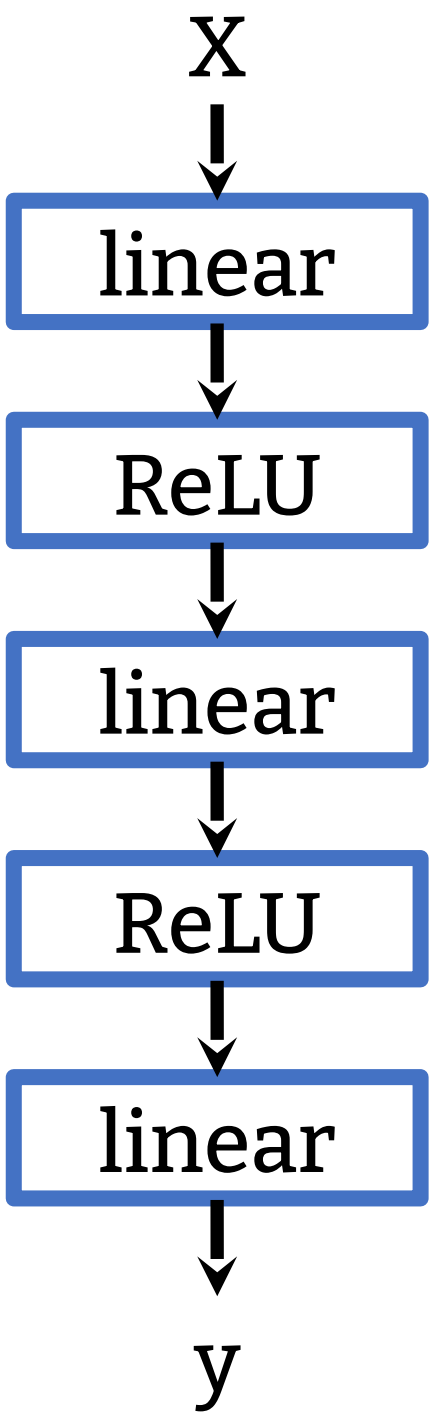
Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

Gradient boosting



MLP

```
class Net(nn.Module):
    def __init__(self, n_input, k1, k2):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(n_input, k1)
        self.fc2 = nn.Linear(k1, k2)
        self.fc3 = nn.Linear(k2, 1)
    def forward(self, x):
        h1 = F.relu( self.fc1(x) )
        h2 = F.relu( self.fc2(h1) )
        return self.fc3(h2)

model = Net( X.shape[1], 200, 200 )
optimizer = optim.Adam( model.parameters() )
loss_fn = nn.MSELoss(reduction='mean')
for epoch in tqdm(range(10_000)):
    optimizer.zero_grad()
    y_hat = model(X_)
    loss = loss_fn( y_hat, y_ )
    loss.backward()
    optimizer.step()
```



MIP monotonicity proof

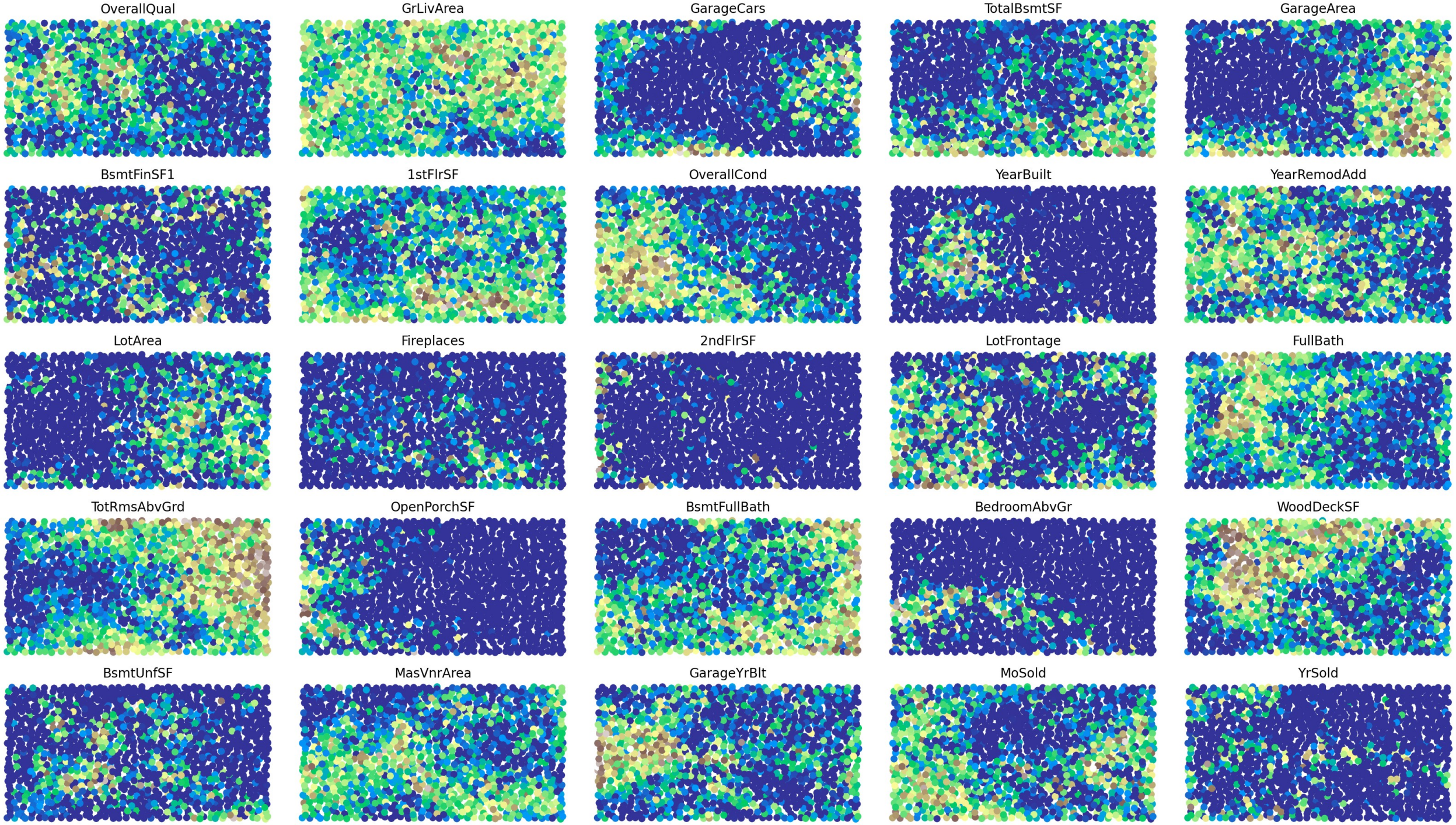
- A neural network with linear layers and ReLU activations is a piecewise linear function: it can be used in a linear program.
- Monotonicity can be formulated as the infeasibility of a linear program.

$$\begin{array}{ll} \text{Find} & x_1, x_2 \in \mathbf{R}^d \\ \text{To maximize} & f(x_1) - f(x_2) - \lambda \sum_i (x_{1i} - x_{2i})_+ \end{array}$$

Certified monotonic neural networks, X. Liu et al. (2020)

Interpretable models

- Some have robust, maintained implementations, but most only have an unmaintained proof-of-concept implementation
- Some are easy to implement from scratch
- Some are not


OverallQual
GrLivArea
GarageCars
TotalBsmtSF
GarageArea
BsmtFinSF1
1stFlrSF
OverallCond
YearBuilt
YearRemodAdd
LotArea
Fireplaces
2ndFlrSF
LotFrontage
FullBath
TotRmsAbvGrd
OpenPorchSF
BsmtFullBath
BedroomAbvGr
WoodDeckSF
BsmtUnfSF
MasVnrArea
GarageYrBlt
MoSold
YrSold

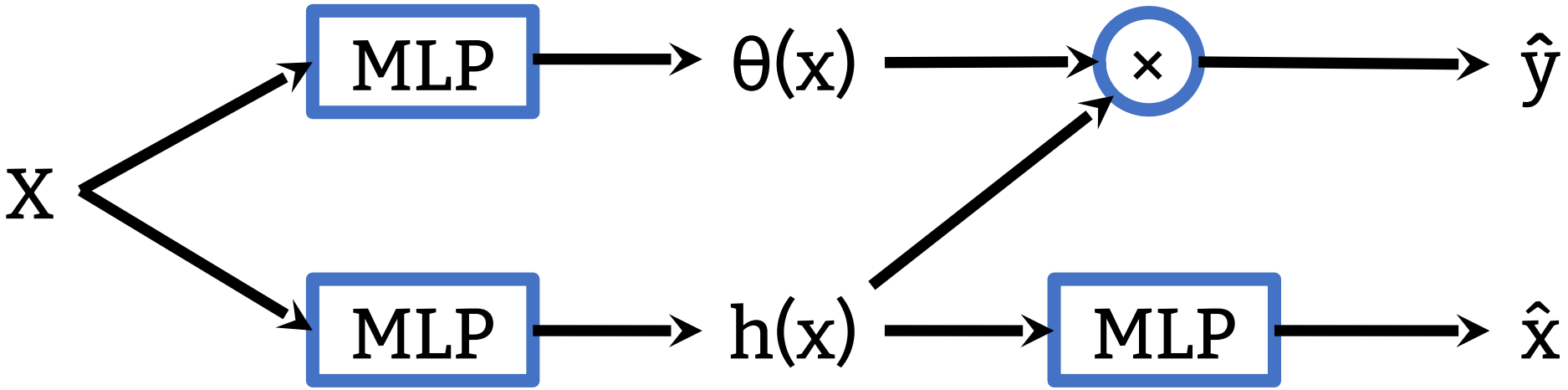
Variable Coefficient Models

- Reconstruction loss: $\|x - \hat{x}\|^2$
- Prediction loss: $\|y - \hat{y}\|^2$
- Penalty: $\left\| \frac{\partial \hat{y}}{\partial x} - \theta \right\|^2$

Self-explaining neural nets

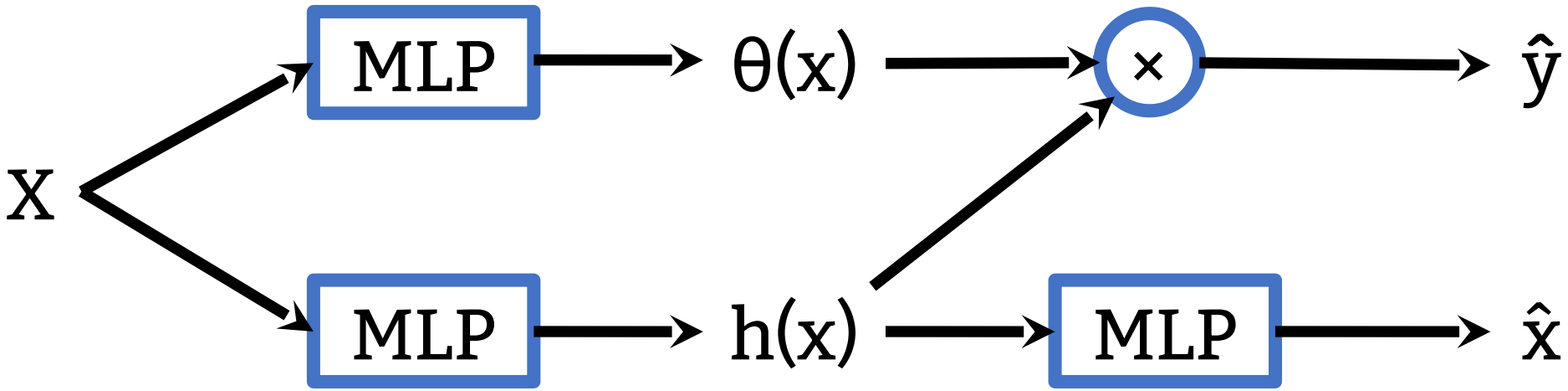
Locally linear model: $f(x) = \sum_i \theta_i(x) h_i(x)$

where θ varies very slowly and *h* comes from an auto-encoder.



Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

Self-explaining neural nets



Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

Examples from the papers

Falling Rule Lists

TABLE 6 A falling rule list from predicting the probability of appendicitis in pediatric patients based on tabular data comprising clinical, laboratory, scoring, and ultrasonography variables.

#	Rule	Probability
1.	Surrounding tissue reaction $=$ *yes* AND Age $\notin [9.3,11.5]$	0.96
2.	Surrounding tissue reaction $=$ *yes* AND Dysuria $=$ *no*	0.94
3.	Pathological lymph nodes $=$ *no* AND Appendix on Ultrasound $=$ *yes*	0.72
4.	Peritonitis $=$ *local* AND Erythrocytes in Urine < 3.0	0.60
5.	C-reactive protein $\in [7.0,31.75]$ AND Alvarado Score $\in [7,10]$	0.60

Note: The risk decreases monotonically across the list.

Table 1: Falling Rule List for bank-full Dataset

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND default=no	THEN success prob. is	0.28	434	1113
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1539
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.23	242	794
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.14	658	4092
ELSE IF	previous ≥ 2 AND education=tertiary	THEN success prob. is	0.13	108	707
ELSE		success prob. is	0.07	2365	31146

	antecedent		prob.
IF	poutcome=success AND default=no	THEN success prob. is	0.65
ELSE IF	$60 \leq$ age < 100 AND default=no	THEN success prob. is	0.28
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.23
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.14
ELSE IF	previous ≥ 2 AND education=tertiary	THEN success prob. is	0.13
ELSE		success prob. is	0.07

	antecedent		probability	positive support	negative support
IF	poutcome=success AND housing=no	THEN success prob. is	0.70	729	311
ELSE IF	poutcome=success	THEN success prob. is	0.53	249	222
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1538
ELSE IF	education=tertiary AND housing=no	THEN success prob. is	0.14	790	4750
ELSE IF	marital=single AND contact=cellular	THEN success prob. is	0.12	648	4754
ELSE IF	$1000 \leq$ balance < 2000 AND housing=no	THEN success prob. is	0.11	135	1061
ELSE IF	campaign=1 AND contact=cellular	THEN success prob. is	0.10	571	4904
ELSE IF	contact=cellular AND loan=no	THEN success prob. is	0.08	587	6800
ELSE		success prob. is	0.04	650	14552

Table 21: Falling rule list for bank-full dataset, trained using the Bayesian approach with 6000 iterations.

	antecedent		probability	positive support	negative support
IF	poutcome=success AND housing=no	THEN success prob. is	0.70	729	311
ELSE IF	poutcome=success AND previous ≥ 2	THEN success prob. is	0.55	185	154
ELSE IF	poutcome=success AND default=no	THEN success prob. is	0.48	64	68
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.25	302	921
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.24	444	1413
ELSE IF	education=tertiary AND housing=no	THEN success prob. is	0.13	671	4435
ELSE		success prob. is	0.07	2468	31590

Table 22: Falling rule list for bank-full dataset, trained using the optimization approach (Algorithm FRL) with 3000 iterations and the positive class weight $w = 7$.

SLIM

TABLE 7 A supersparse linear integer model for predicting the risk of appendicitis in pediatric patients based on tabular data comprising clinical, laboratory, scoring, and ultrasonography variables (Roig Aparicio et al., 2021).

Condition	Score
Peritonitis = *generalized*	6
Appendix diameter = 9–17 mm	6
Appendix diameter = 5.9–9.0 mm	5
Appendix on ultrasound = *yes*	4
Peritonitis = *local*	2

Note: Every feature has an integral coefficient attached to it.

Fig. 1: SLIM scoring system for the `breastcancer` dataset.

Clump Thickness (1 to 10)		.....
Uniformity of Cell Size (1 to 10)	+	.....
Bare Nuclei (1 to 10)	+	.....
	−	10
Total	=	.....

$$
\begin{aligned}
\text{Score} = \; & 0.24 \times \text{ClumpThickness} + 0.15 \times \text{UniformityOfCellSize} \\
& + 0.20 \times \text{UniformityOfCellShape} + 0.10 \times \text{MarginalAdhesion} \\
& + 0.34 \times \text{BareNuclei} + 0.13 \times \text{NormalNucleoli} - 4.98 \\
\text{Class} = \; & \text{sign}(\text{Score}).
\end{aligned}
$$

Fig. 2: Decision tree induced by the SLIM scoring system for the mammo dataset.

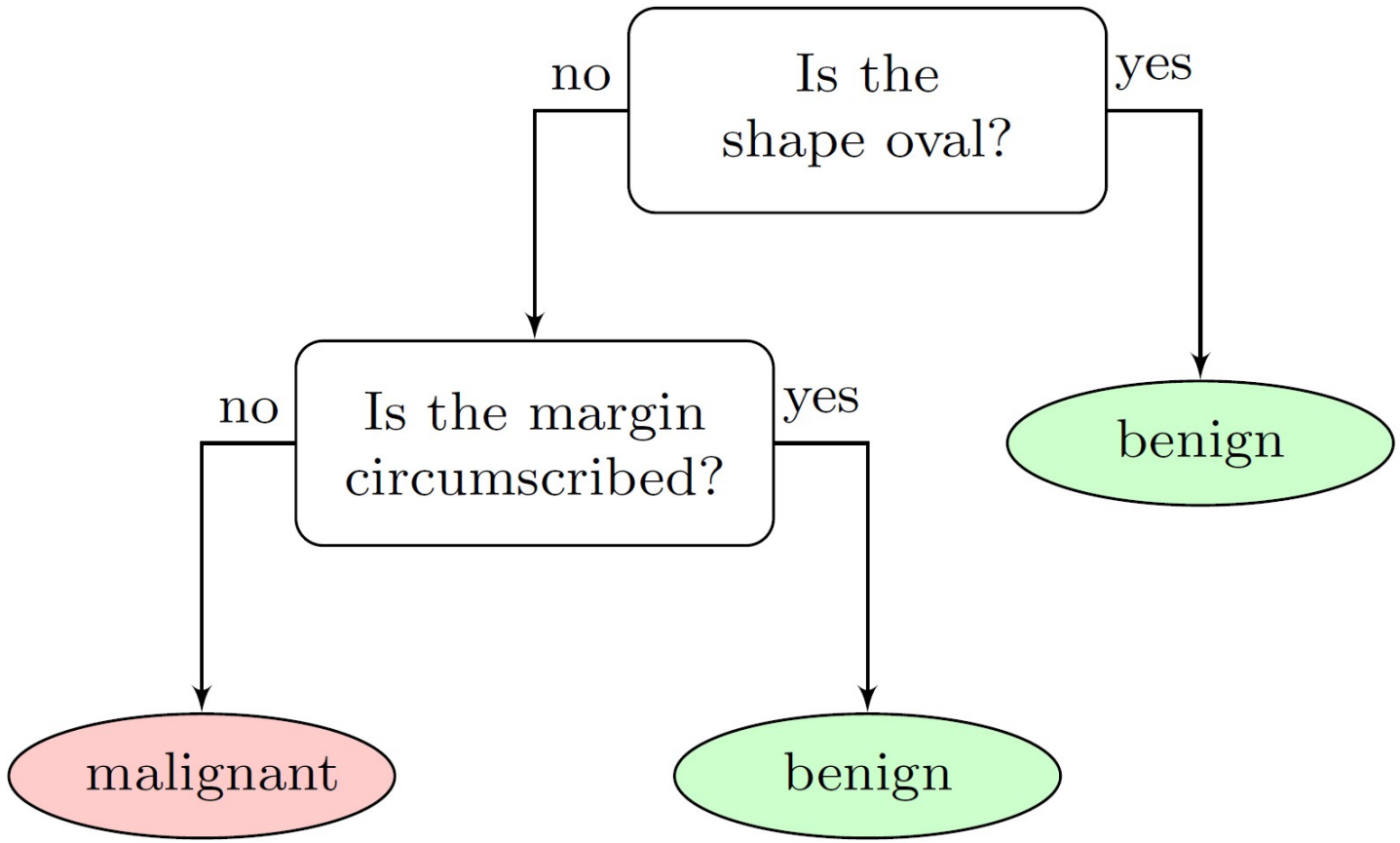


Fig. 3: SLIM scoring system for the `violentcrime` dataset.

1) Does the person have a mental health problem?	(10 points)	.....
2) Has the person ever used or threatened to use a weapon?	(5 points)	.....
3) Has the person ever shot or stabbed someone?	(5 points)	.....
4) Has the person ever stolen something worth over $50?	(5 points)	.....
5) Is the person male and distanced from his dad?	(5 points)	.....
6) Does the person not have a dad or stepdad?	(1 point)	.....
7) Is the person male and not have a dad or stepdad?	(1 point)	.....
8) Does the person not have a mom or stepmom?	(1 point)	.....
9) Is the person male and not have a mom or stepmom?	(1 point)	.....
Sum points from 1 to 9	**Total A**	.....
10) Is the person female and not have a dad or stepdad?	(10 points)	.....
11) Does the person have college plans?	(5 points)	.....
12) Is the person employed?	(1 point)	.....
13) Is the person in school and employed?	(1 point)	.....
14) Likelihood to use child welfare system.	(1-4 points)	.....
Sum points from 10 to 14	**Total B**	.....
Subtract Total B from Total A	**Total C**	.....

PREDICT MUSHROOM IS POISONOUS IF SCORE > 3

1.	$spore\_print\_color = green$	4 points		$\cdots\cdots$
2.	$stalk\_surface\_above\_ring = grooves$	2 points	$+$	$\cdots\cdots$
3.	$population = clustered$	2 points	$+$	$\cdots\cdots$
4.	$gill\_size = broad$	-2 points	$+$	$\cdots\cdots$
5.	$odor \in \{none, almond, anise\}$	-4 points	$+$	$\cdots\cdots$
	ADD POINTS FROM ROWS 1–5	**SCORE**	$=$	$\cdots\cdots$

Fig. 9: SLIM scoring system for `mushroom`. This model has a 10-CV mean test error of $0.0 \pm 0.0\%$.

	10.86	*spore_print_color = green*	+	4.49	*gill_size = narrow*	+	4.29 *odor = foul*
+	2.73	*stalk_surface_below_ring = scaly*	+	2.60	*stalk_surface_above_ring = grooves*	+	2.38 *population = clustered*
+	0.85	*spore_print_color = white*	+	0.44	*stalk_root = bulbous*	+	0.43 *gill_spacing = close*
+	0.38	*cap_color = white*	+	0.01	*stalk_color_below_ring = yellow*	−	8.61 *odor = anise*
−	8.61	*odor = almond*	−	8.51	*odor = none*	−	0.53 *cap_surface = fibrous*
−	0.25	*population = solitary*	−	0.21	*stalk_surface_below_ring = fibrous*	−	0.09 *spore_print_color = brown*
−	0.00	*cap_shape = convex*	−	0.00	*gill_spacing = crowded*	−	0.00 *gill_size = broad*
+	0.25						

Fig. 10: Lasso score function for `mushroom`. This model has a 10-CV mean test error of $0.0 \pm 0.0\%$.

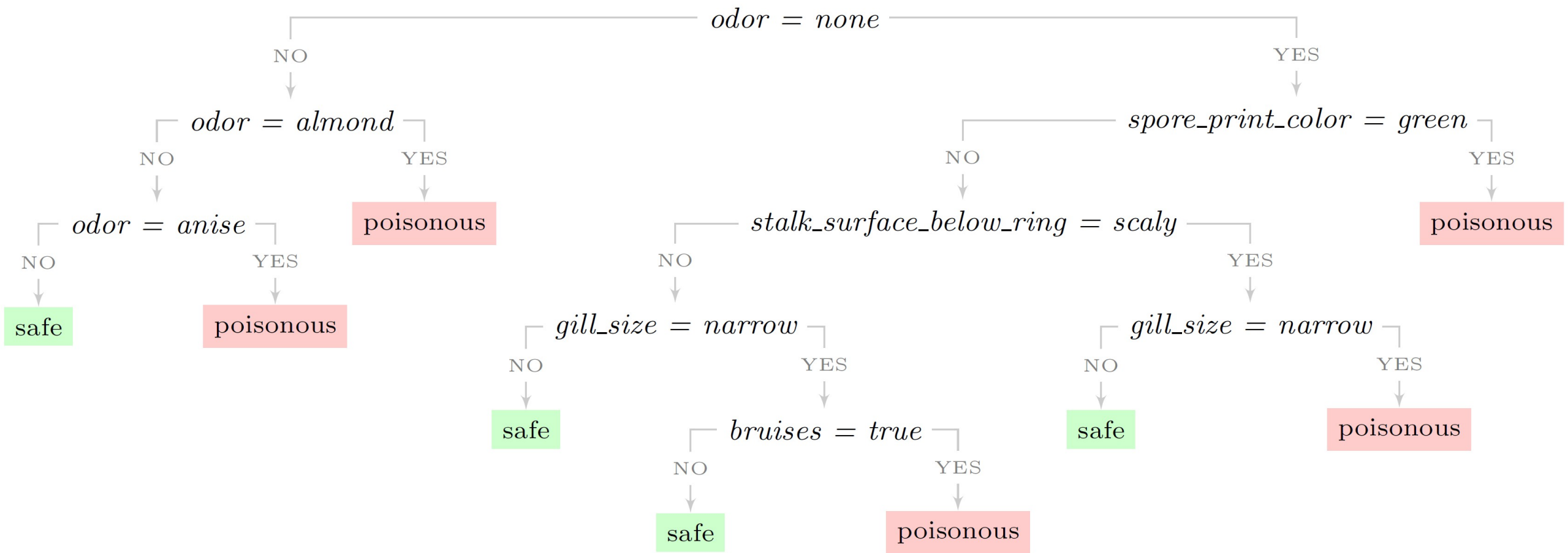


Fig. 11: C5.0 decision tree for `mushroom`. This model has a 10-CV mean test error of 0.0±0.0%.

Rule			Confidence	Support	Lift
$odor = none \ \wedge \ gill\_size \neq narrow \ \wedge \ spore\_print\_color \neq green$	$\Longrightarrow$	safe	1.000	3216	1.9
$bruises = false \ \wedge \ odor = none \ \wedge \ stalk\_surface\_below\_ring \neq scaly$	$\Longrightarrow$	safe	0.999	1440	1.9
$odor = almond$	$\Longrightarrow$	safe	0.998	400	1.9
$odor = anise$	$\Longrightarrow$	safe	0.998	400	1.9
$odor \neq almond \ \wedge \ odor \neq anise \ \wedge \ odor \neq none$	$\Longrightarrow$	poisonous	1.000	3796	2.1
$spore\_print\_color = green$	$\Longrightarrow$	poisonous	0.986	72	2.9
$gill\_size = narrow \ \wedge \ stalk\_surface\_below\_ring = scaly$	$\Longrightarrow$	poisonous	0.976	40	2.0

Fig. 12: C5.0 rule list for `mushroom`. This model has a 10-CV mean test error of $0.0 \pm 0.0\%$.

PREDICT TUMOR IS BENIGN
IF AT LEAST 5 OF THE FOLLOWING 8 RULES ARE TRUE

$$UniformityOfCellSize \geq 3$$
$$UniformityOfCellShape \geq 3$$
$$MarginalAdhesion \geq 3$$
$$SingleEpithelialCellSize \geq 3$$
$$BareNuclei \geq 3$$
$$NormalNucleoli \geq 3$$
$$BlandChromatin \geq 3$$
$$Mitoses \geq 3$$

Fig. 15: M-of-N rule table for the `breastcancer` dataset for $C_0 = 0.9/NP$. This model has 8 rules and a 10-CV mean test error of $4.8 \pm 2.5\%$. We trained this model with binary rules $h_{i,j} := \mathbb{1}[x_{i,j} \geq 3]$.

Self-explaining neural net

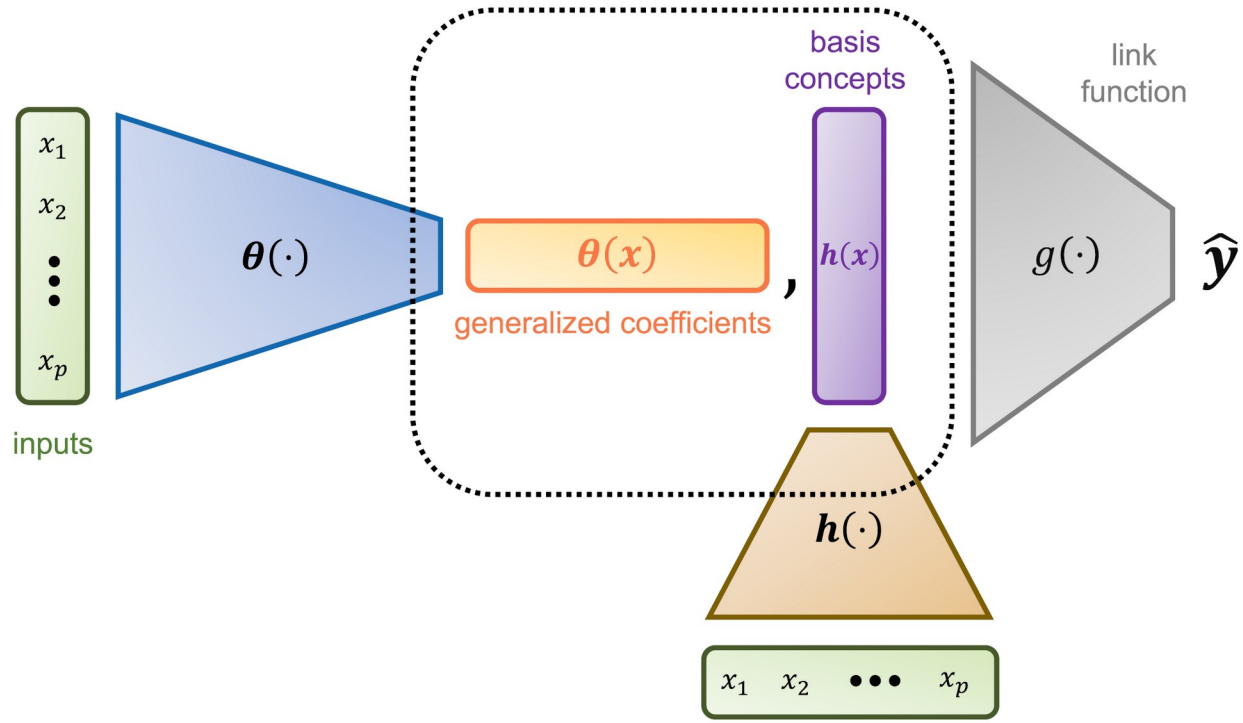
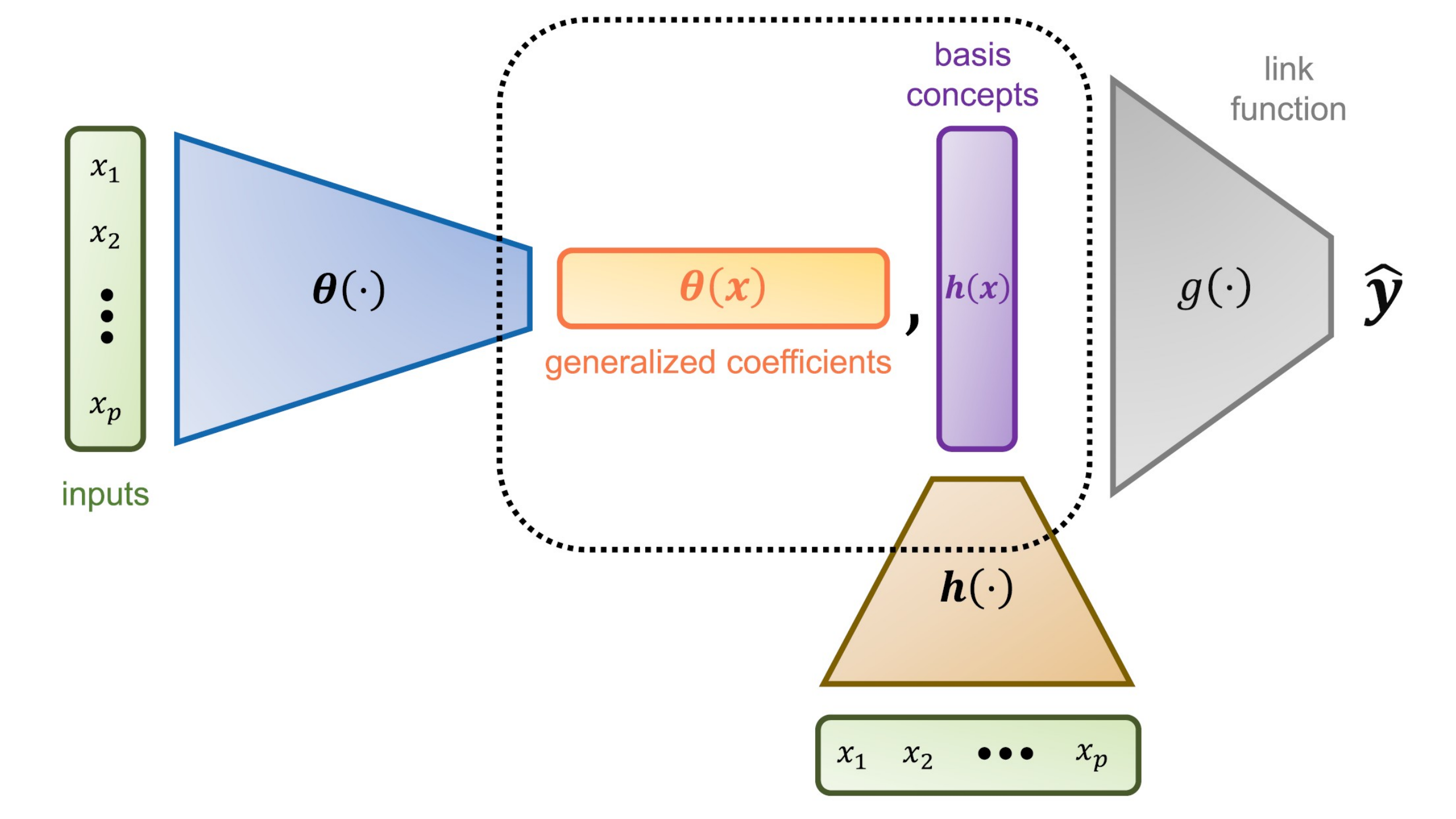


FIGURE 4 A schematic depiction of a self-explaining neural network model. Input variables $x_1,...,x_p$ are mapped to generalized coefficients and interpretable basis concepts by neural networks $\boldsymbol{\theta}(\cdot)$ and $\boldsymbol{h}(\cdot)$, respectively. Generalized coefficients and basis concepts are then combined by an interpretable link function $g(\cdot)$ into a predicted value $\hat{\boldsymbol{y}}$.


basis
concepts
link
function
x_1
x_2
x_p
$\boldsymbol{\theta}(\cdot)$
$\boldsymbol{\theta}(\boldsymbol{x})$
,
$\boldsymbol{h}(\boldsymbol{x})$
$g(\cdot)$
$\hat{\boldsymbol{y}}$
generalized coefficients
inputs
$\boldsymbol{h}(\cdot)$
x_1 x_2 ••• x_p

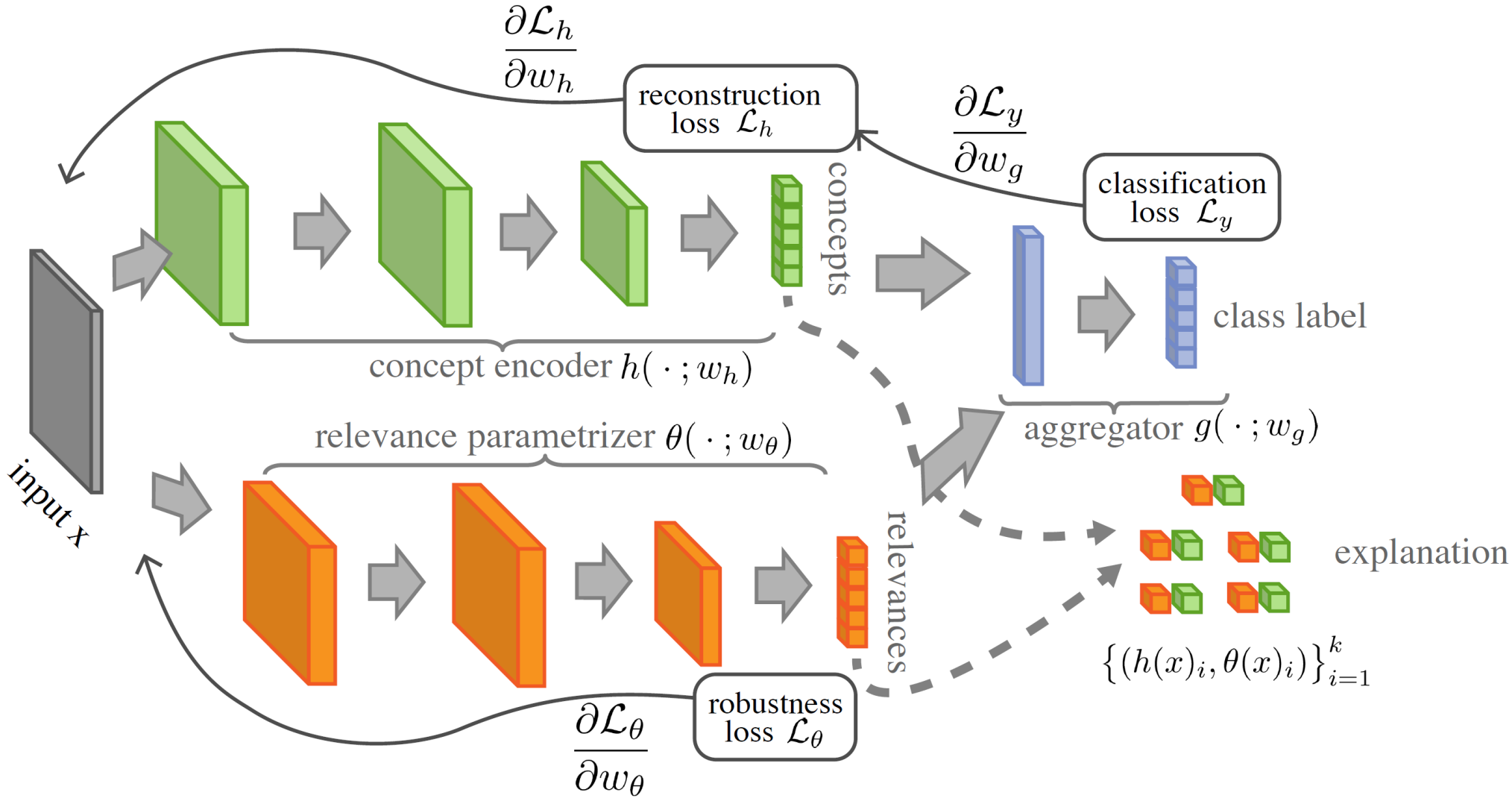


Figure 1: A SENN consists of three components: a **concept encoder** (green) that transforms the input into a small set of interpretable basis features; an **input-dependent parametrizer** (orange) that generates relevance scores; and an **aggregation function** that combines to produce a prediction. The robustness loss on the parametrizer encourages the full model to behave locally as a linear function on $h(x)$ with parameters $\theta(x)$, yielding immediate interpretation of both concepts and relevances.

LRP

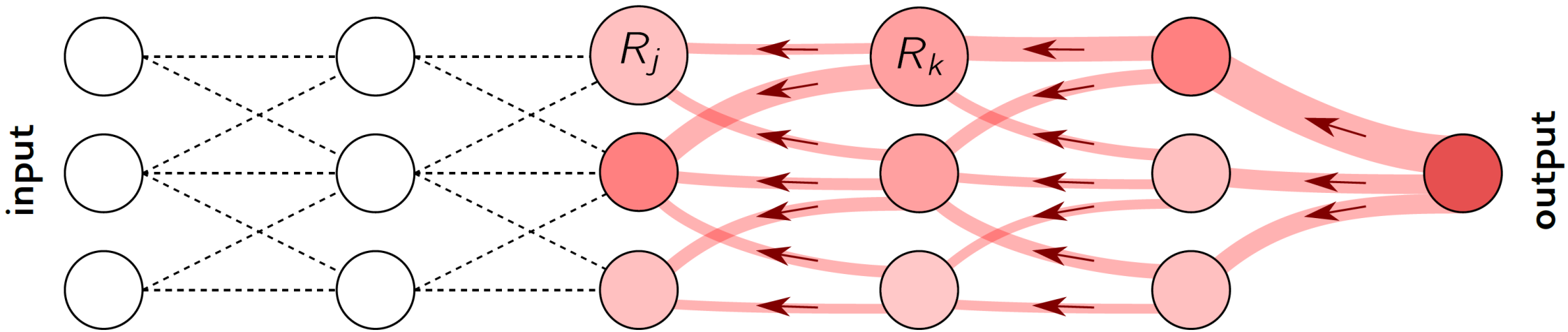


Fig. 10.2. Illustration of the LRP procedure. Each neuron redistributes to the lower layer as much as it has received from the higher layer.

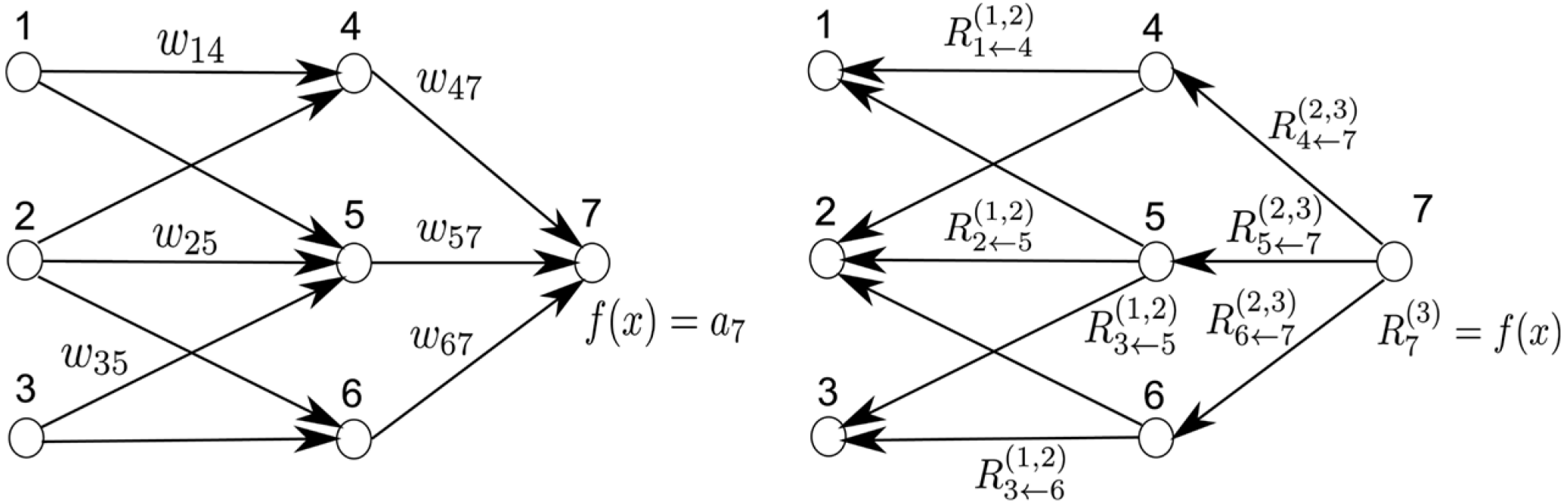


Fig 2. Left: A neural network-shaped classifier during prediction time. w_{ij} are connection weights. a_i is the activation of neuron *i*. Right: The neural network-shaped classifier during layer-wise relevance computation time. $R_i^{(l)}$ is the relevance of neuron *i* which is to be computed. In order to facilitate the computation of $R_i^{(l)}$ we introduce messages $R_{i\leftarrow j}^{(l,l+1)}$. $R_{i\leftarrow j}^{(l,l+1)}$ are messages which need to be computed such that the layer-wise relevance in Eq (2) is conserved. The messages are sent from a neuron *i* to its input neurons *j* via the connections used for classification, e.g. 2 is an input neuron for neurons 4, 5, 6. Neuron 3 is an input neuron for 5, 6. Neurons 4, 5, 6 are the input for neuron 7.

Symbolic Regression

Name	Law	Early Citation
Hubble's law	$v = H_0 D$	[79]
Kepler's Third Law	$P^2 \propto a^3$	[81]
Newton's law of universal gravitation	$\mathbf{F} = G\frac{m_1 m_2}{r^2}\hat{r}$	[82]
Planck's law	$B = \frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/k_B T}-1}$	[83]
Leavitt's Law	$M = \alpha \log_{10}(P) + \delta$	[84]
Schechter function	$n = \frac{\phi_*}{L_*}(\frac{L}{L_*})^\alpha e^{-\frac{L}{L_*}}$	[85]
Bode's law	$a = 0.4 + 0.3(2^n)$	[86]
Ideal gas law	$P = \frac{nRT}{V}$	[87]
Rydberg formula	$\frac{1}{\lambda} = R_H(\frac{1}{n_1^2} - \frac{1}{n_2^2})$	[88]

Table 2: Expressions in the EmpiricalBench and associated with the datasets in fig. 5. Each of these expressions was originally empirically discovered.