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)

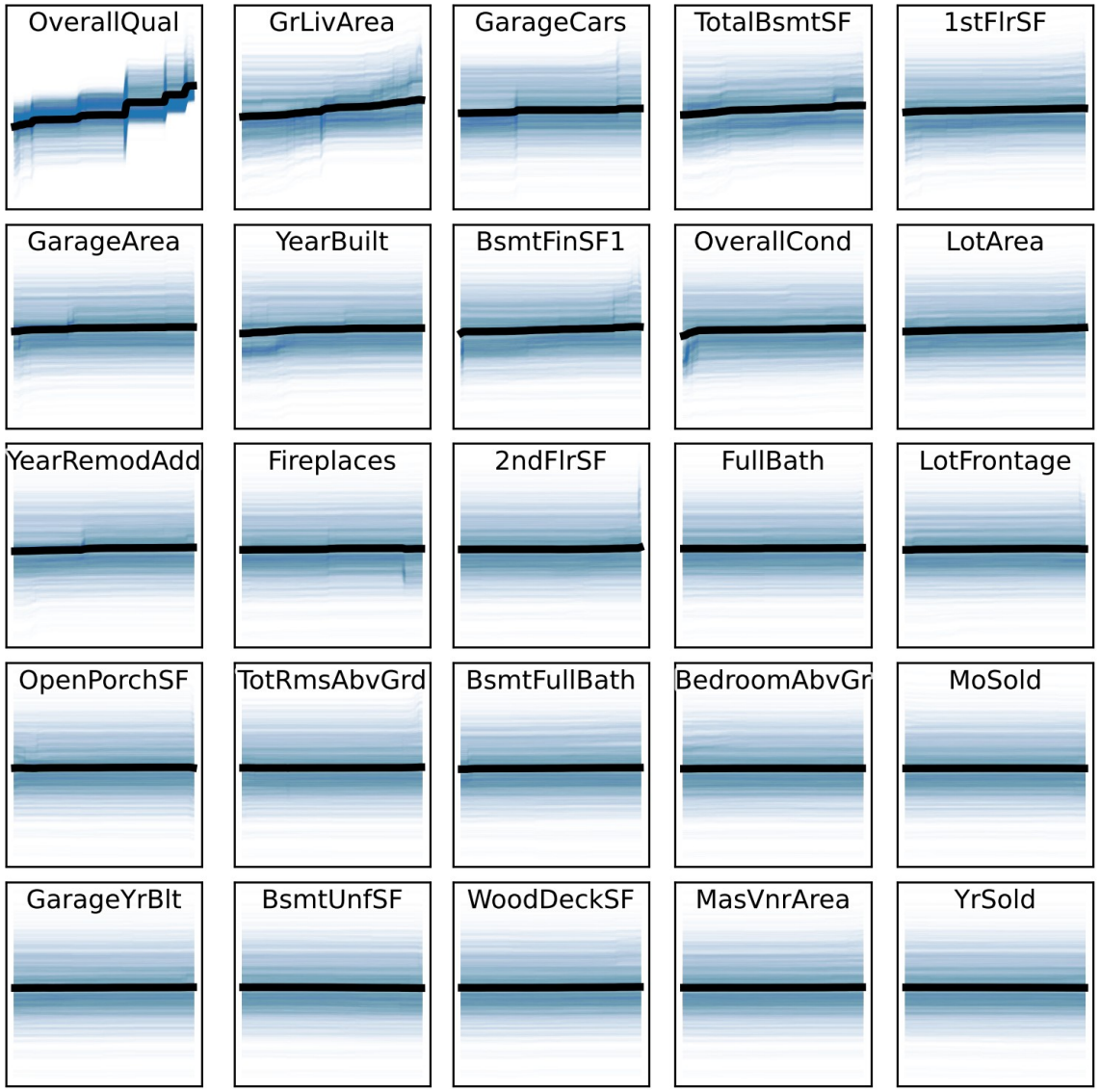
After: Explainable AI

- 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

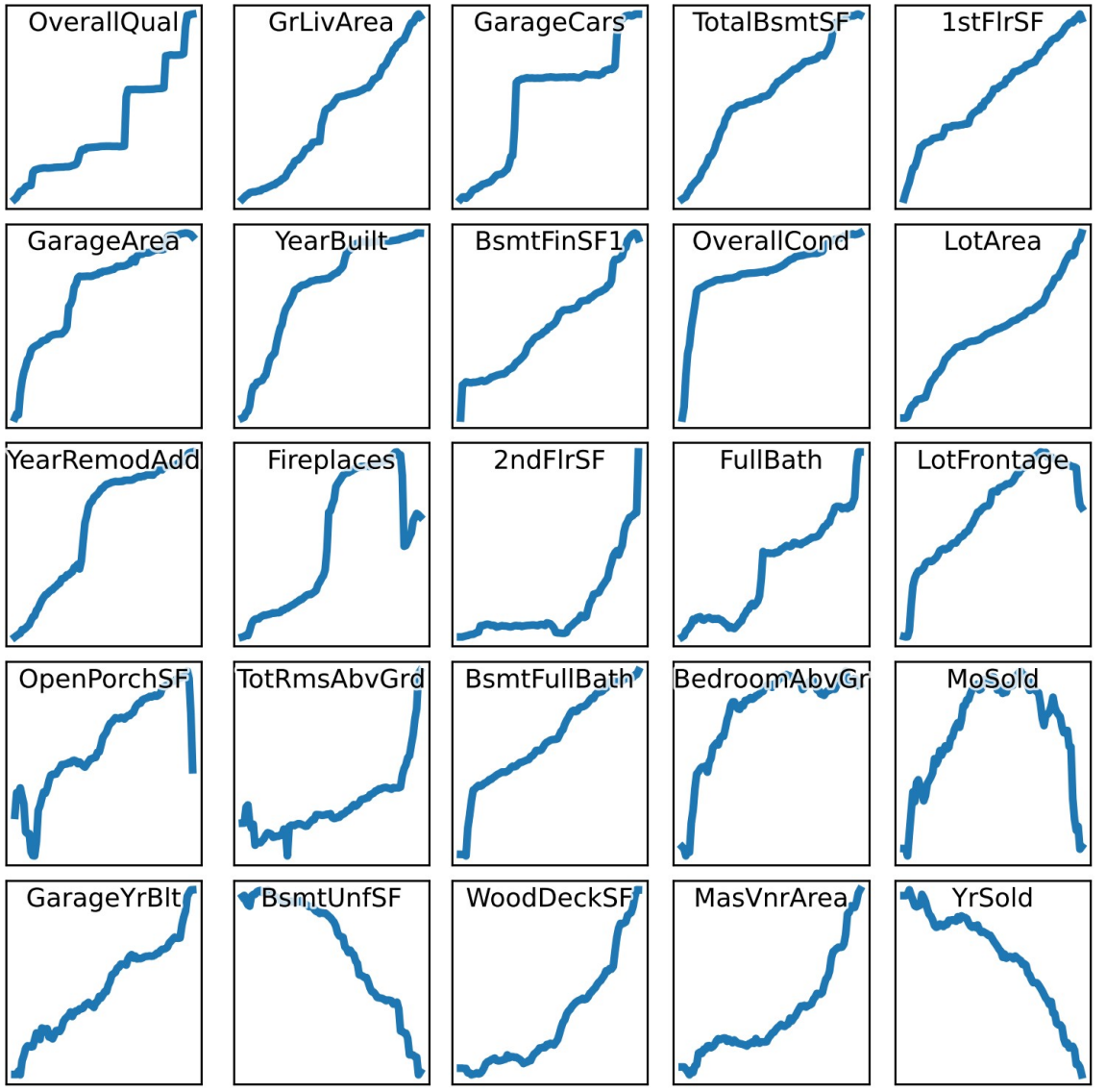
- For each observation i, how does the output change if we change variable j (while keeping the others constant)?
- Average of those curves



Interpretable Machine Learning, C. Molnar (2020)

Partial Dependency Plot

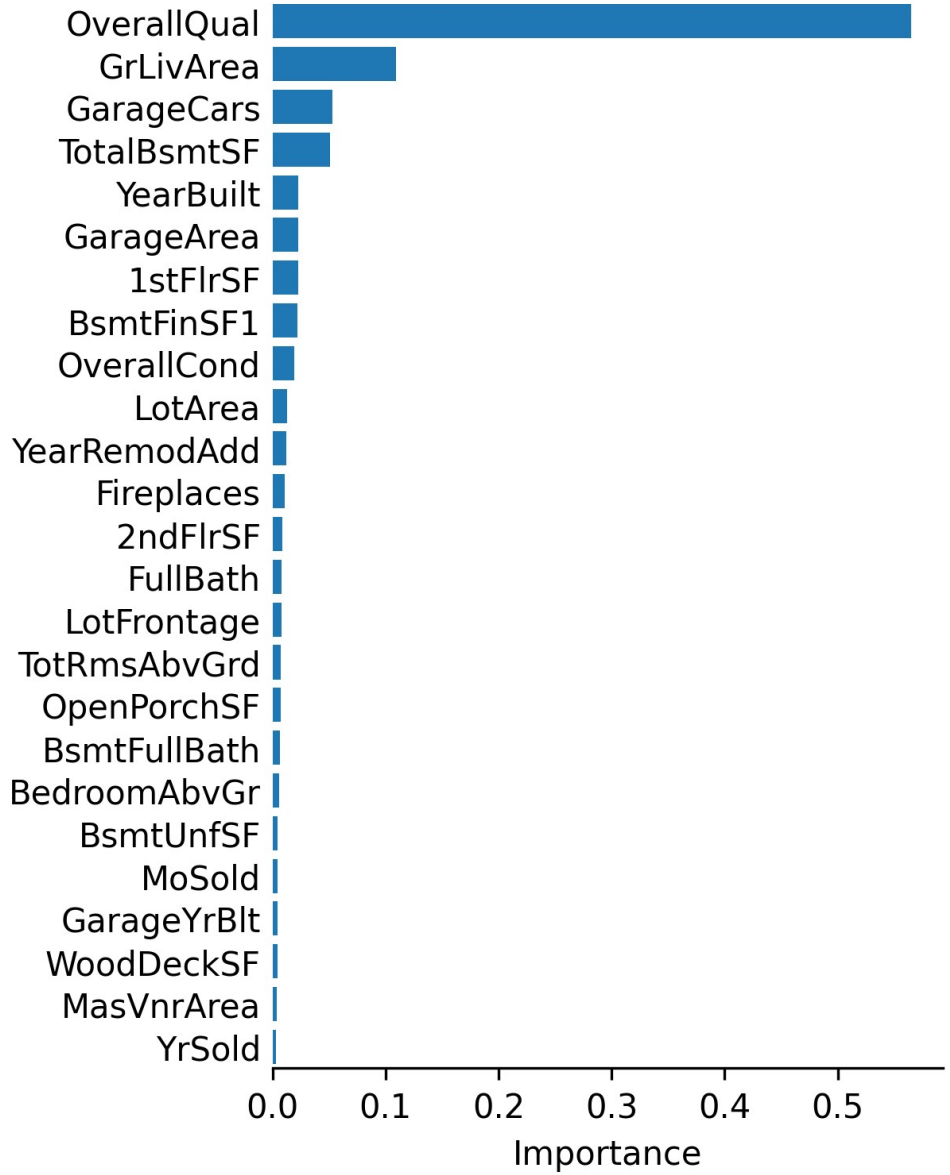
- For each observation i, how does the output change if we change variable j (while keeping the others constant)?
- Average of those curves



Interpretable Machine Learning, C. Molnar (2020)

Feature Importance

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



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)

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, GA^{2}M, AIM

Deep learning models

- Monotonicity: activation functions, gradient penalty
- Sparsity
- Varying coefficient models
- Mixture of experts

Linear Models

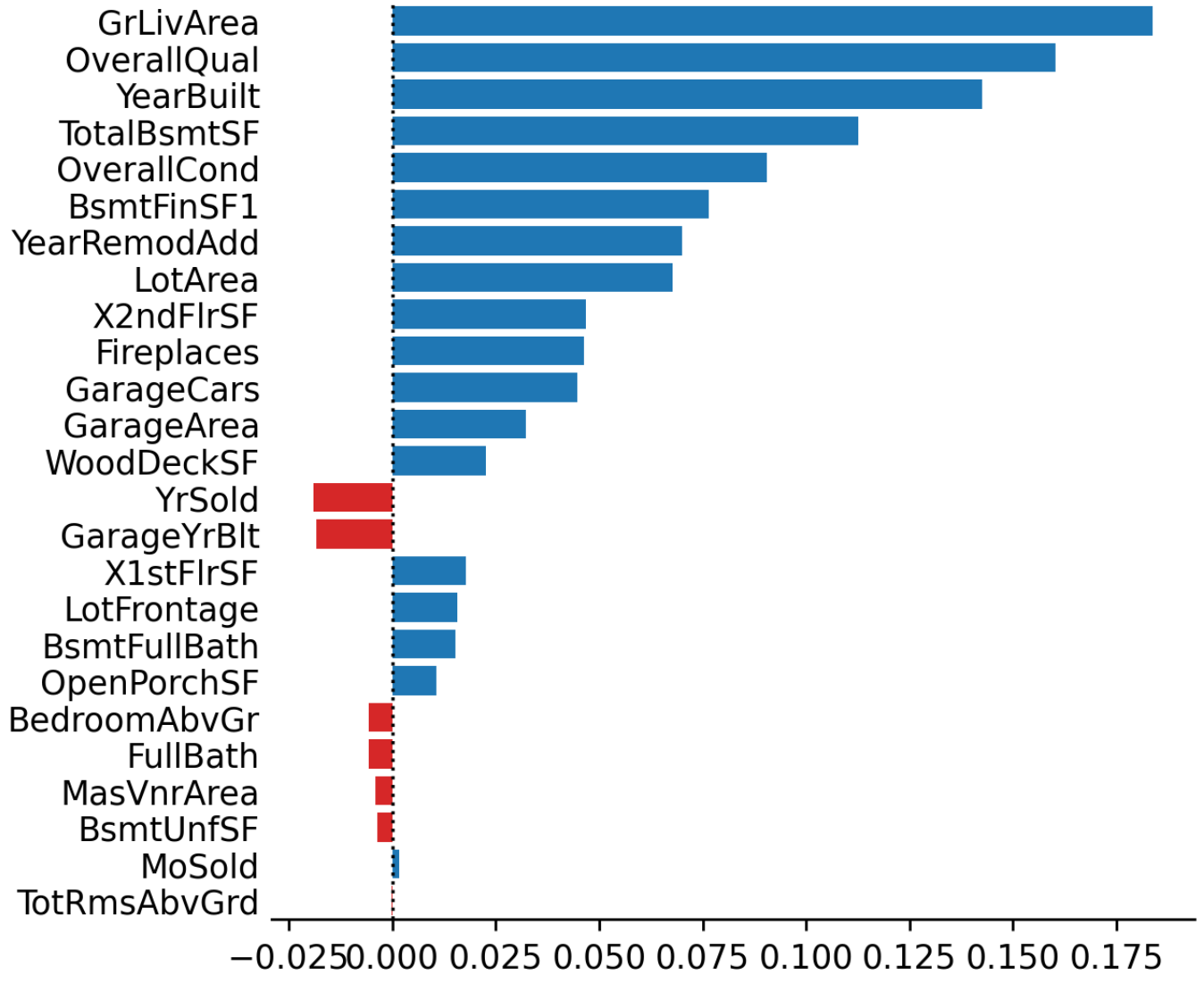
Linear models

- Lasso, elastic net
- Fused lasso, group lasso, graph sparsity
- Sign constraints
- Supersparse integer linear model

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

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

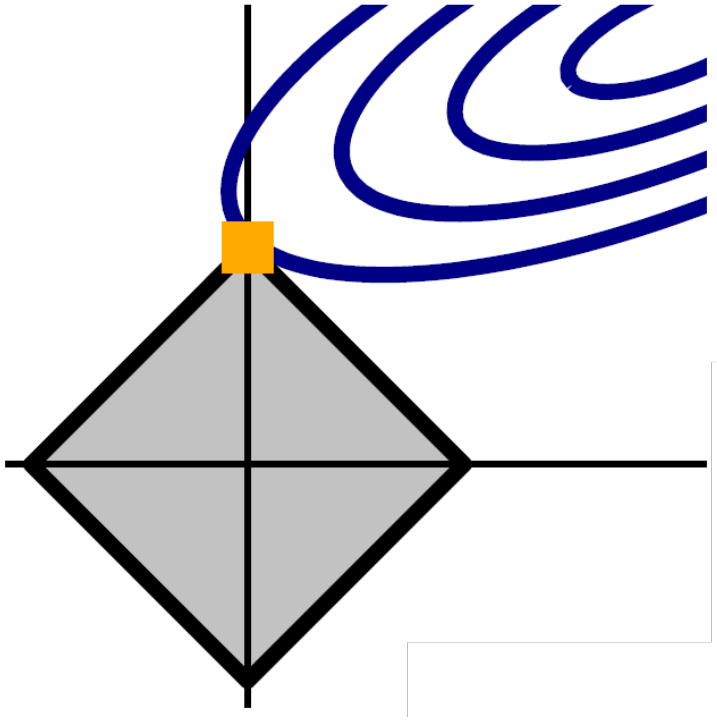
```
import cvxpy as cp
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - ( intercept + X.values @ w.T )
objective = cp.sum_squares( residuals ) / n + alpha * cp.norm1(w)
prob = cp.Problem( cp.Minimize(objective), [] )
result = prob.solve()
w = pd.Series( w.value, index = X.columns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

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

Lasso: sparsity

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

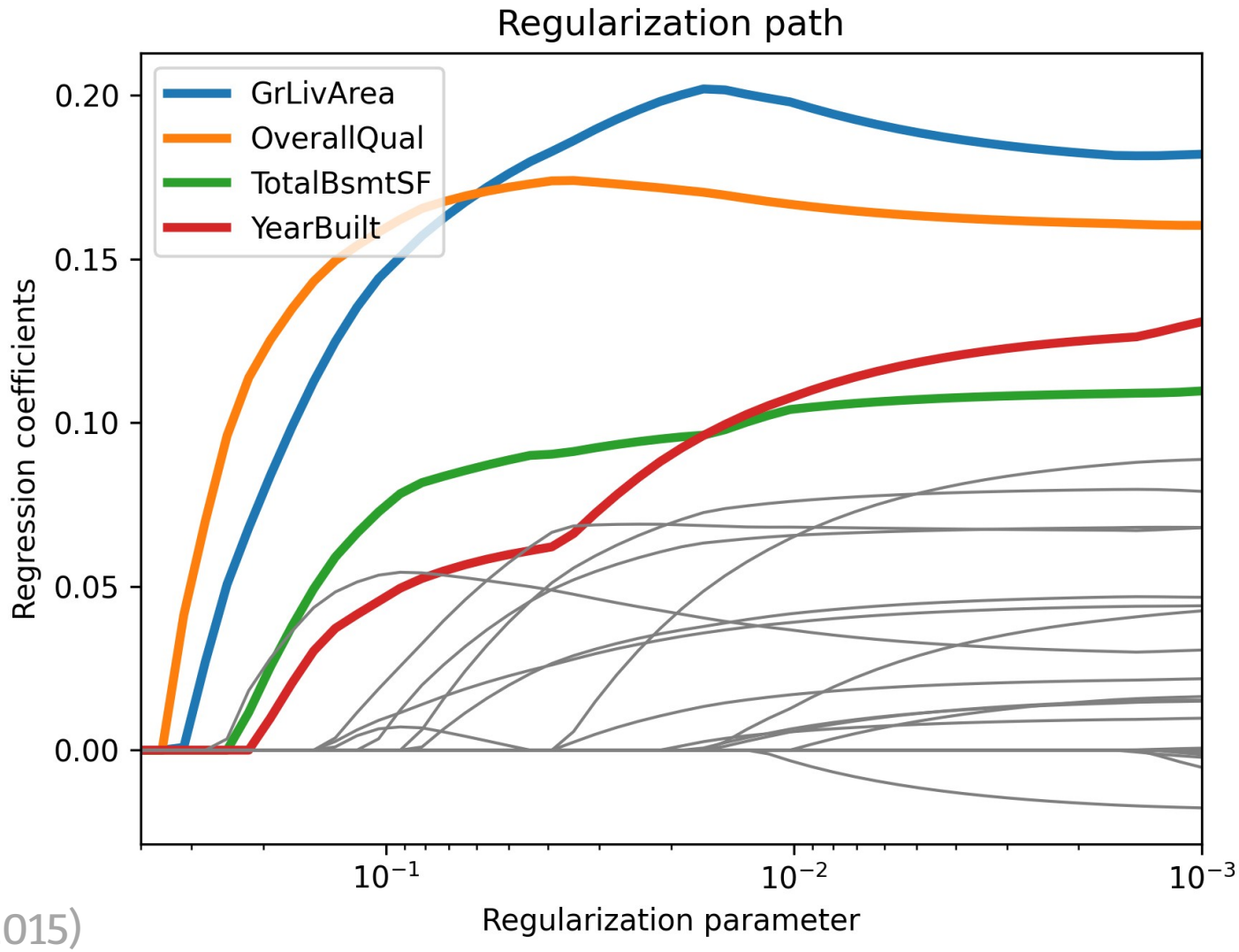
$$\begin{array}{ll} \text{Find} & w \in \mathbf{R}^k \\ \text{To minimize} & \|Xw - y\|_2^2 \\ \text{Such that} & \|w\|_1 \leqslant c \end{array}$$



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

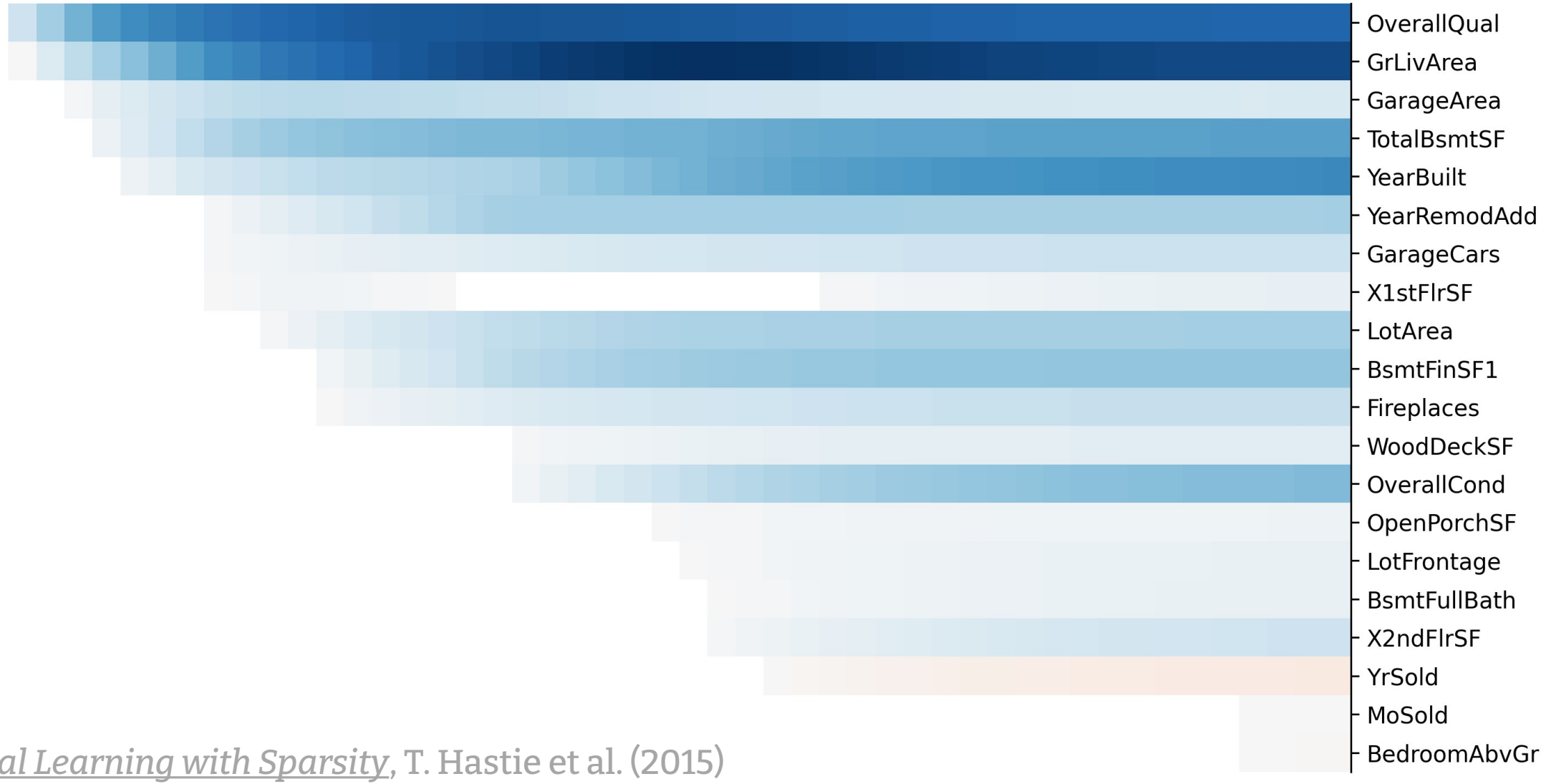
Lasso: regularization path

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)

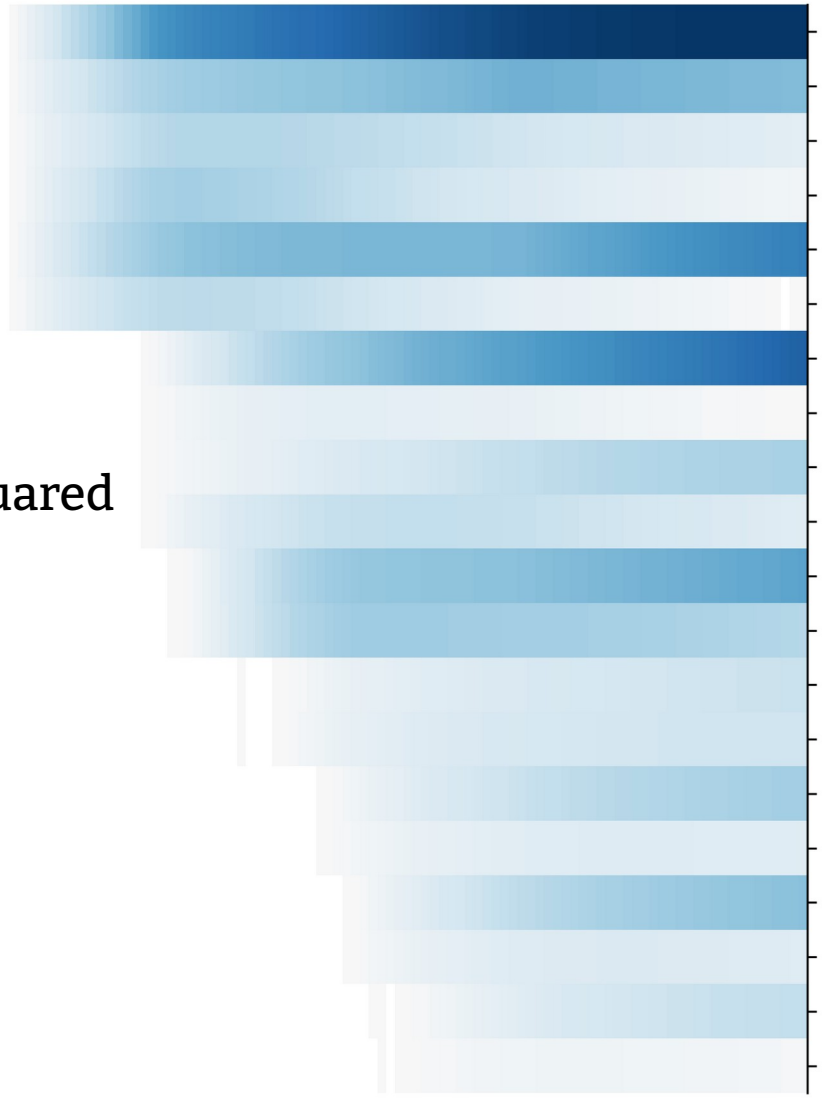
Group lasso

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



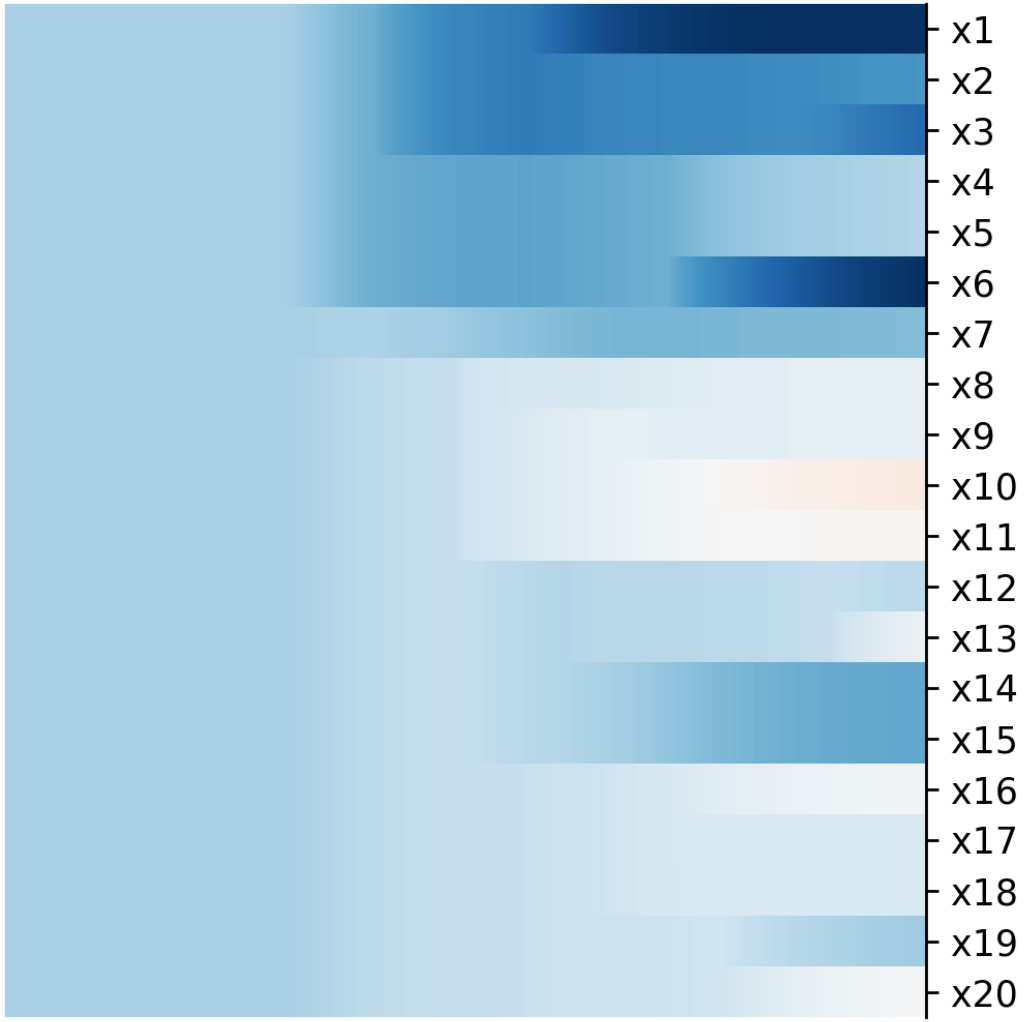
```
# Group lasso (often used for the rows of a matrix)
w = cp.Variable(k)
intercept = cp.Variable(1)
residuals = y.values - ( intercept + X.values @ w.T )
objective = cp.sum_squares( residuals ) / n
for i in np.unique(c):
    objective += alpha * cp.norm2( cp.multiply( (c==i), w ) )
prob = cp.Problem(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

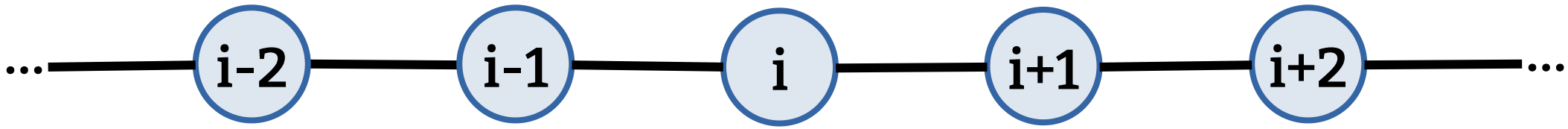


Statistical Learning with Sparsity, T. Hastie et al. (2015)

Fused lasso

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

To minimize $\|Xw - y\|_2^2 + \lambda \sum_i |w_{i+1} - w_i|$

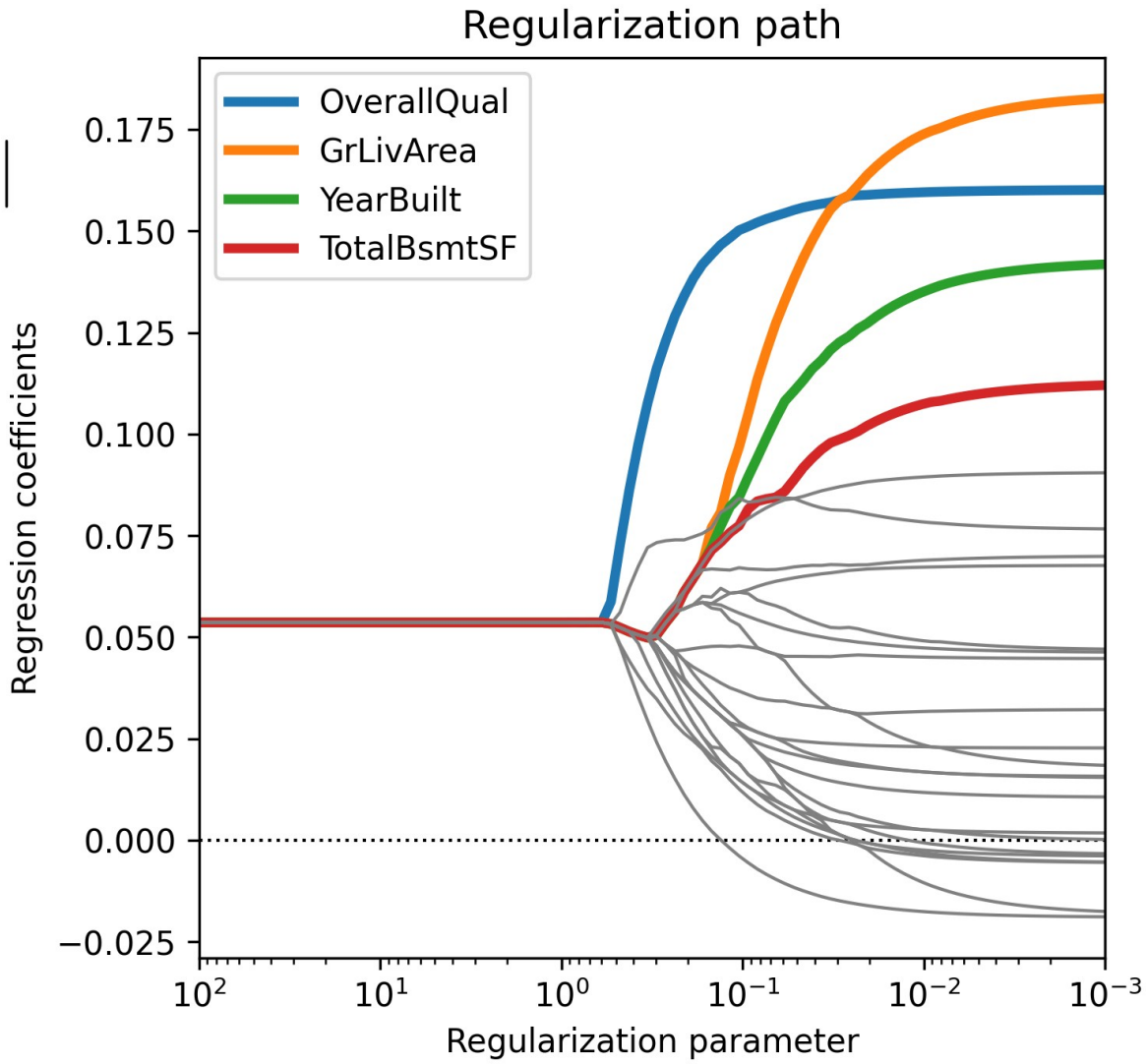


Statistical Learning with Sparsity, T. Hastie et al. (2015)

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
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 )
```



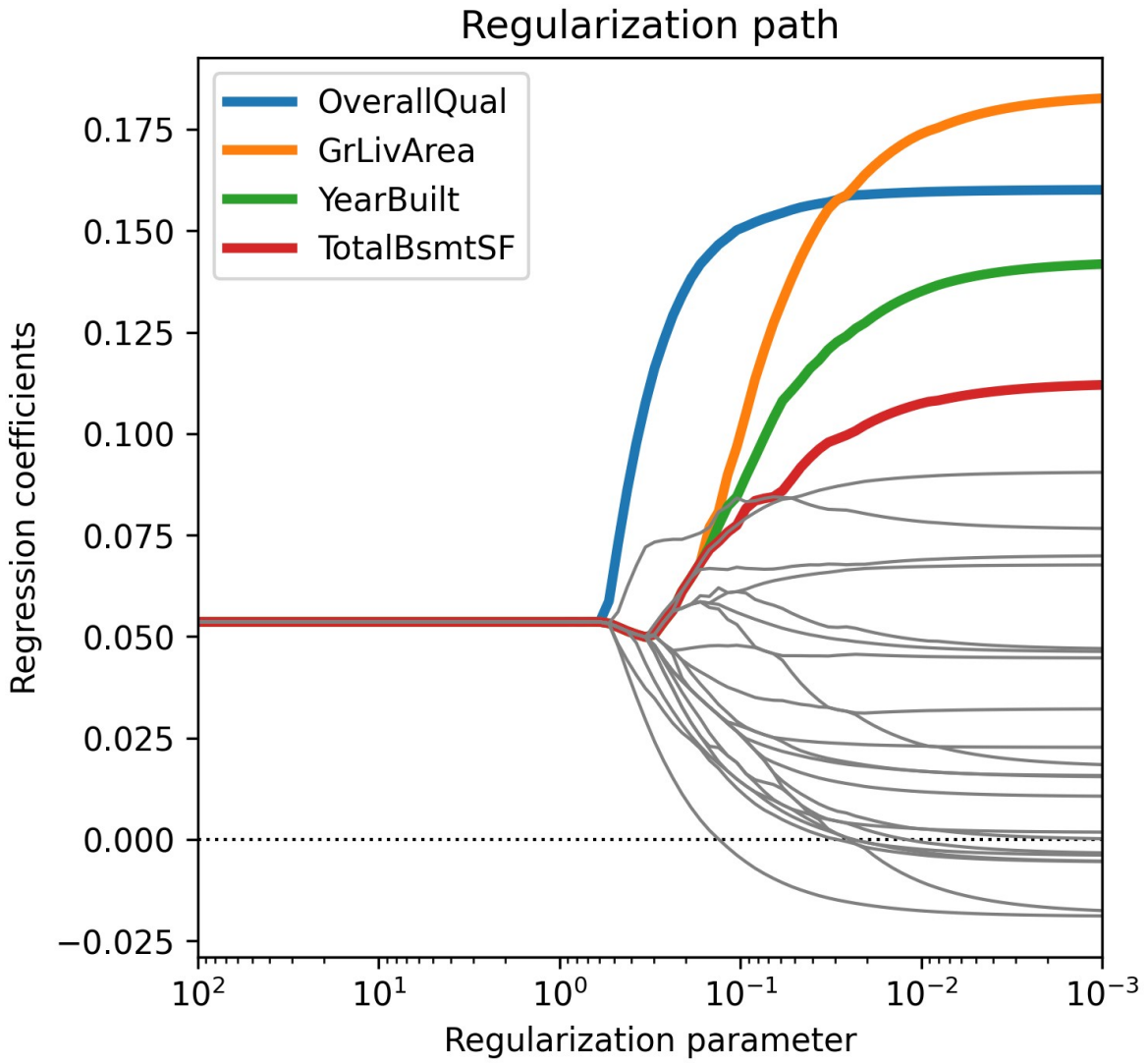
Statistical Learning with Sparsity, T. Hastie et al. (2015)

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 )
```

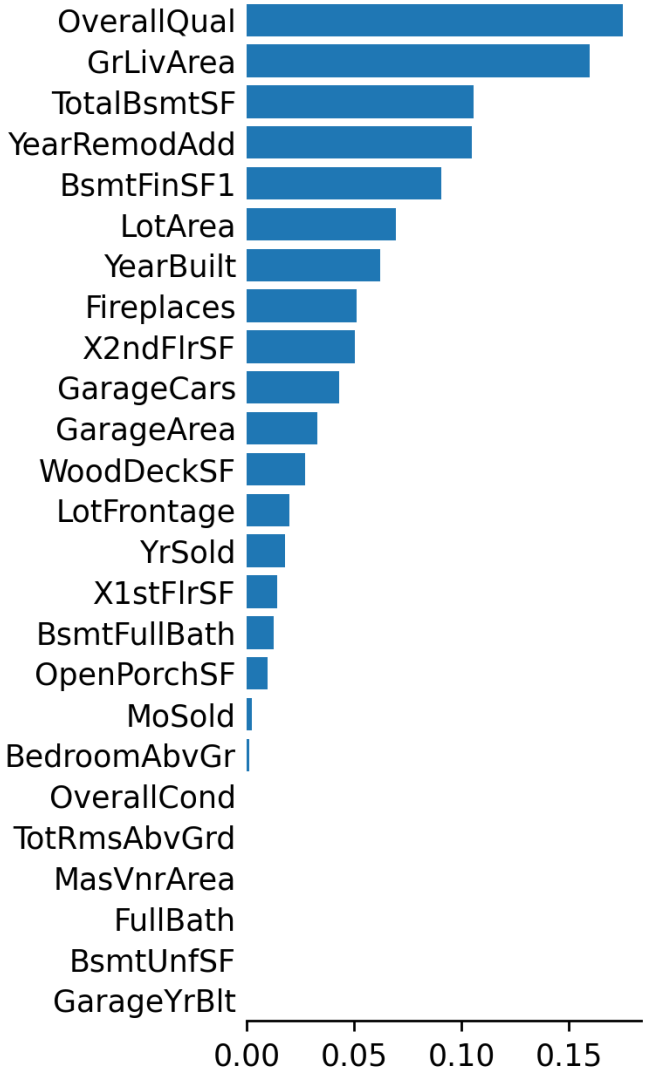
Statistical Learning with Sparsity, T. Hastie et al. (2015)



Sign constraints

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

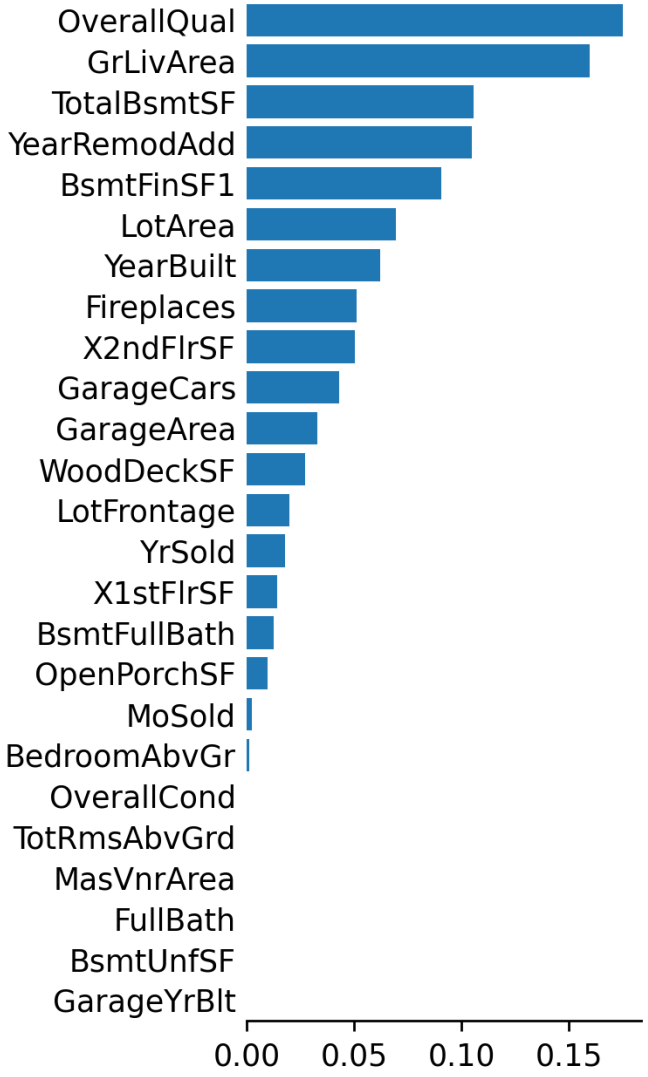
```
# Linear regression with sign constraints
model = LinearRegression( positive = True )
model.fit(X, y)
yhat = model.predict(X)
```



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

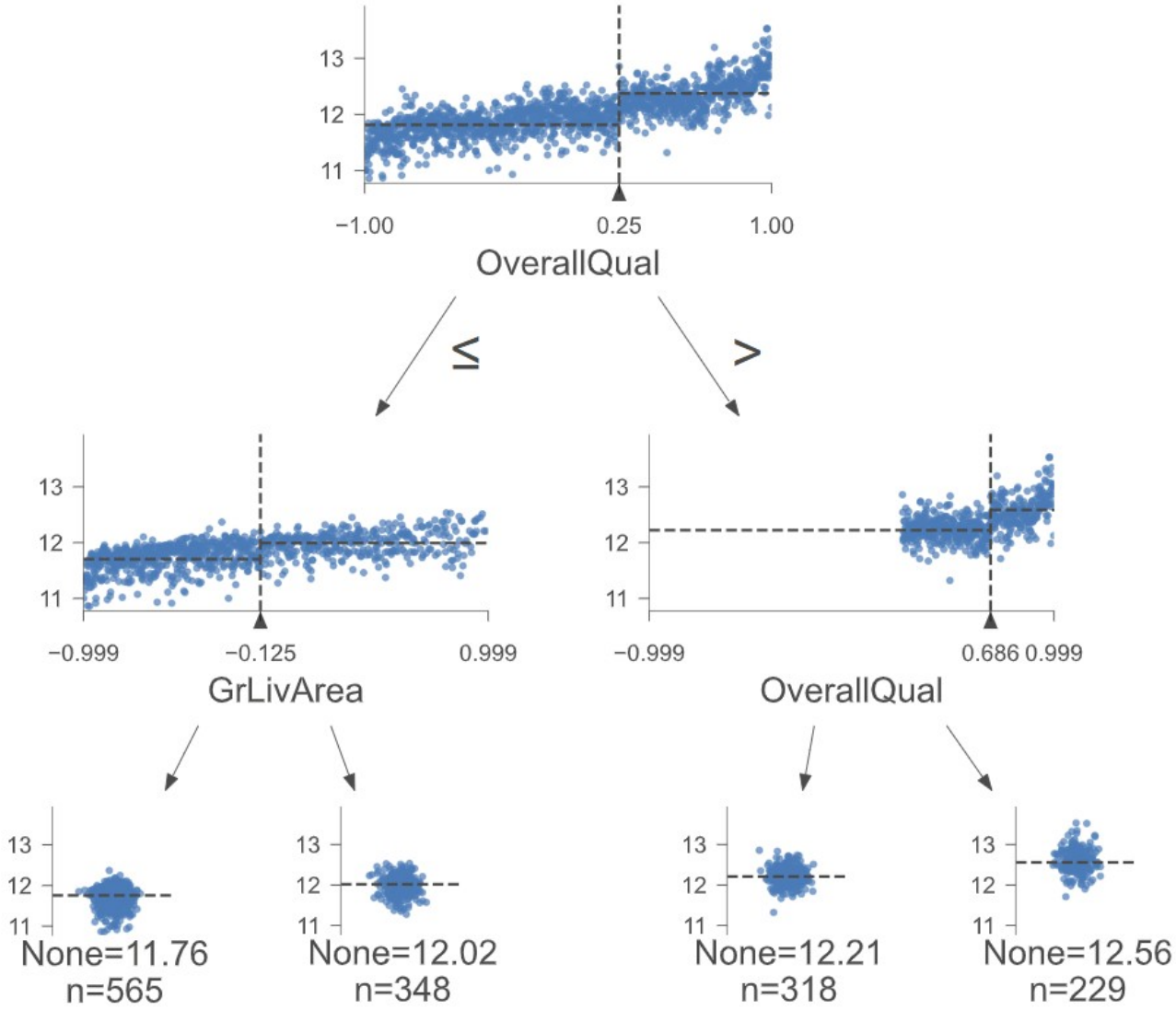
Supersparse linear integer model for interpretable classification, B. Ustun et al. (2013)

Machine Learning

Machine Learning

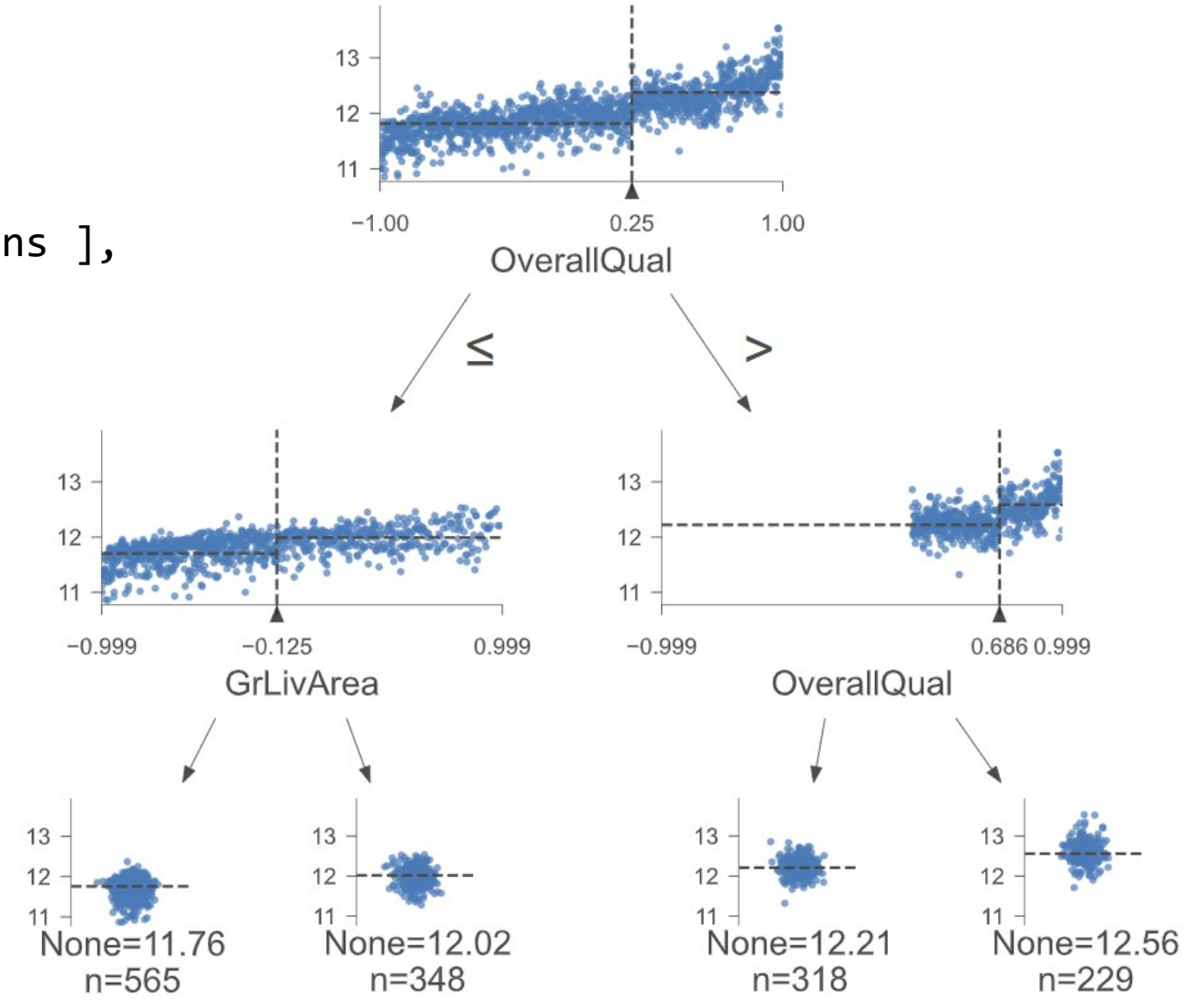
- Decision trees, falling rule lists
- Monotonic decision trees
- Monotonic gradient boosting
- Generalized additive models (GAM)
- GAM with interactions
- GAM boosting
- Additive index models (AIM)
- Symbolic regression

Decision tree

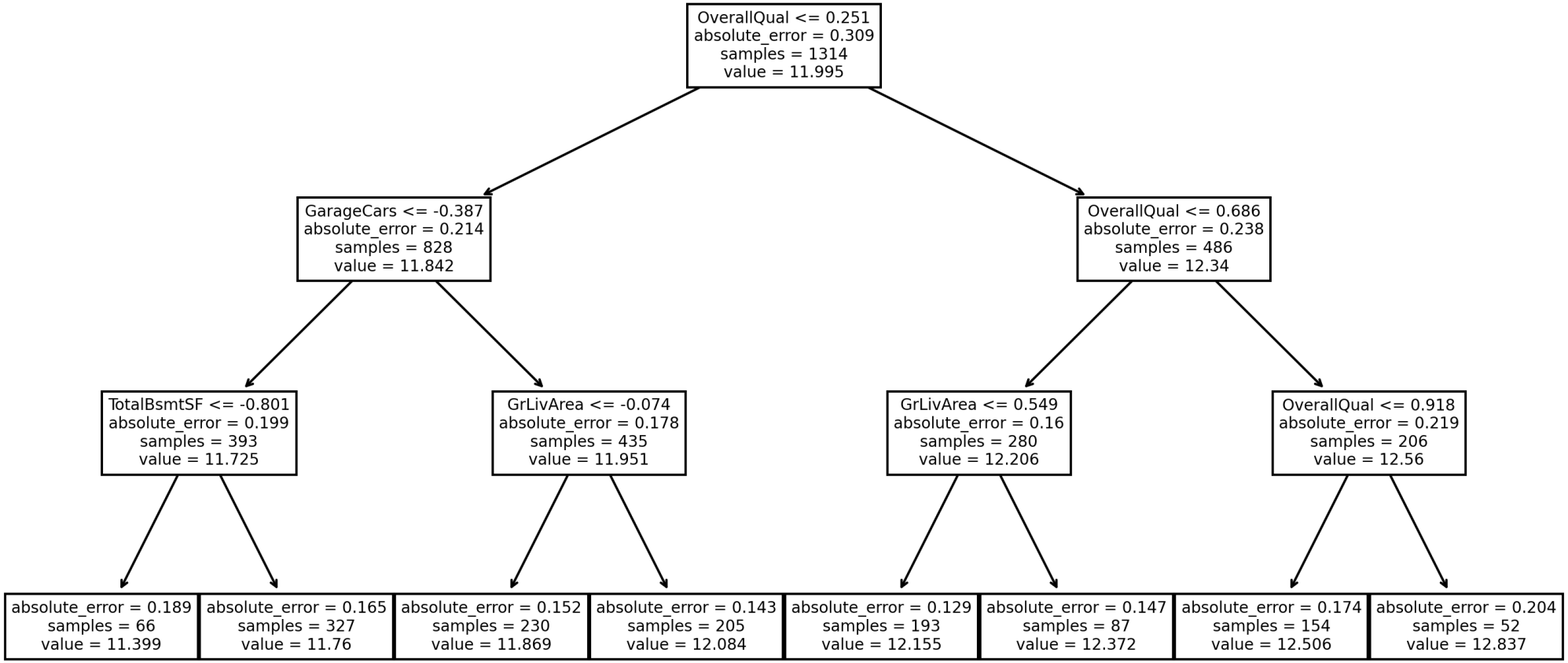


Monotonic decision tree

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

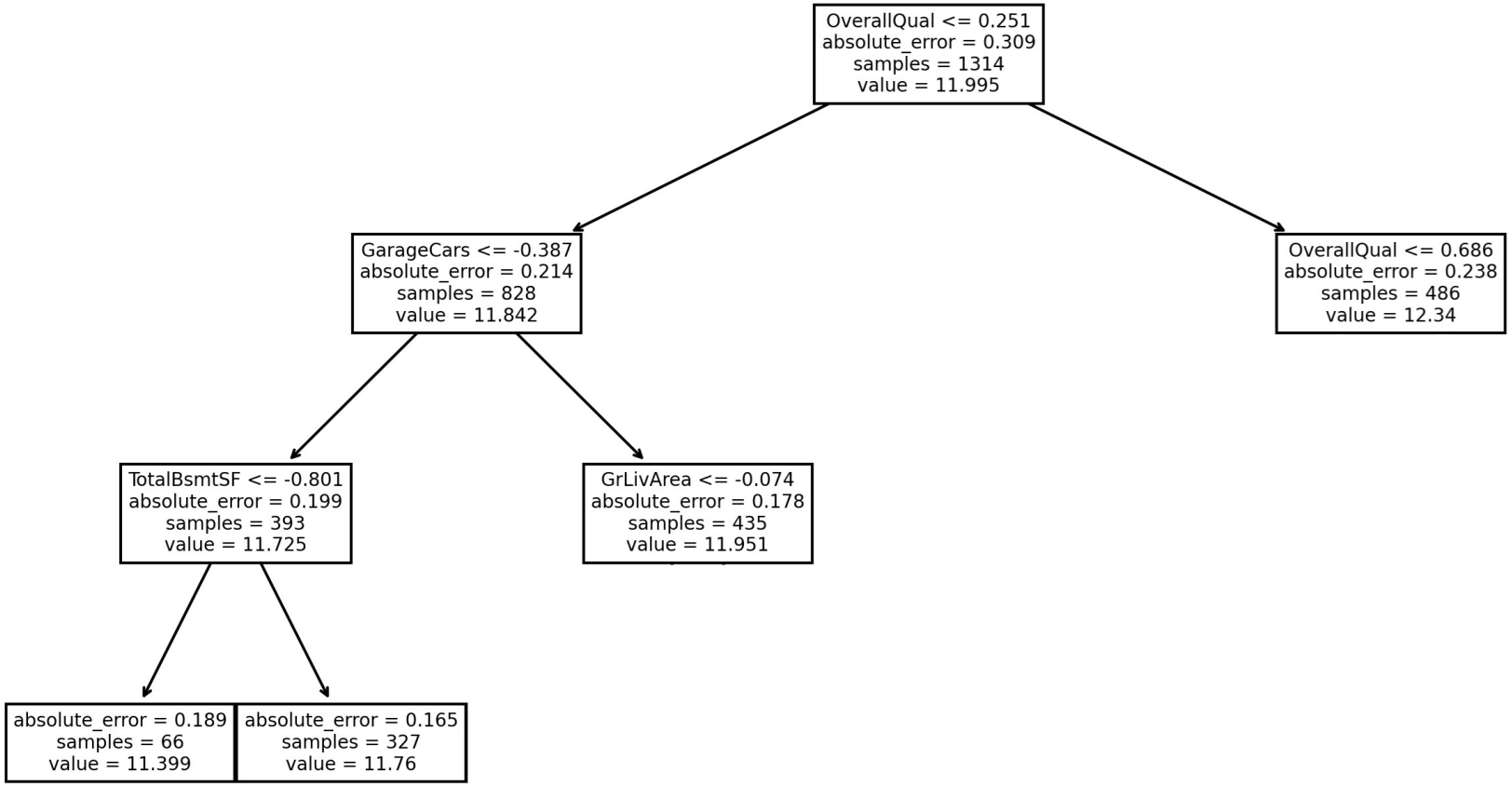


Falling rule lists



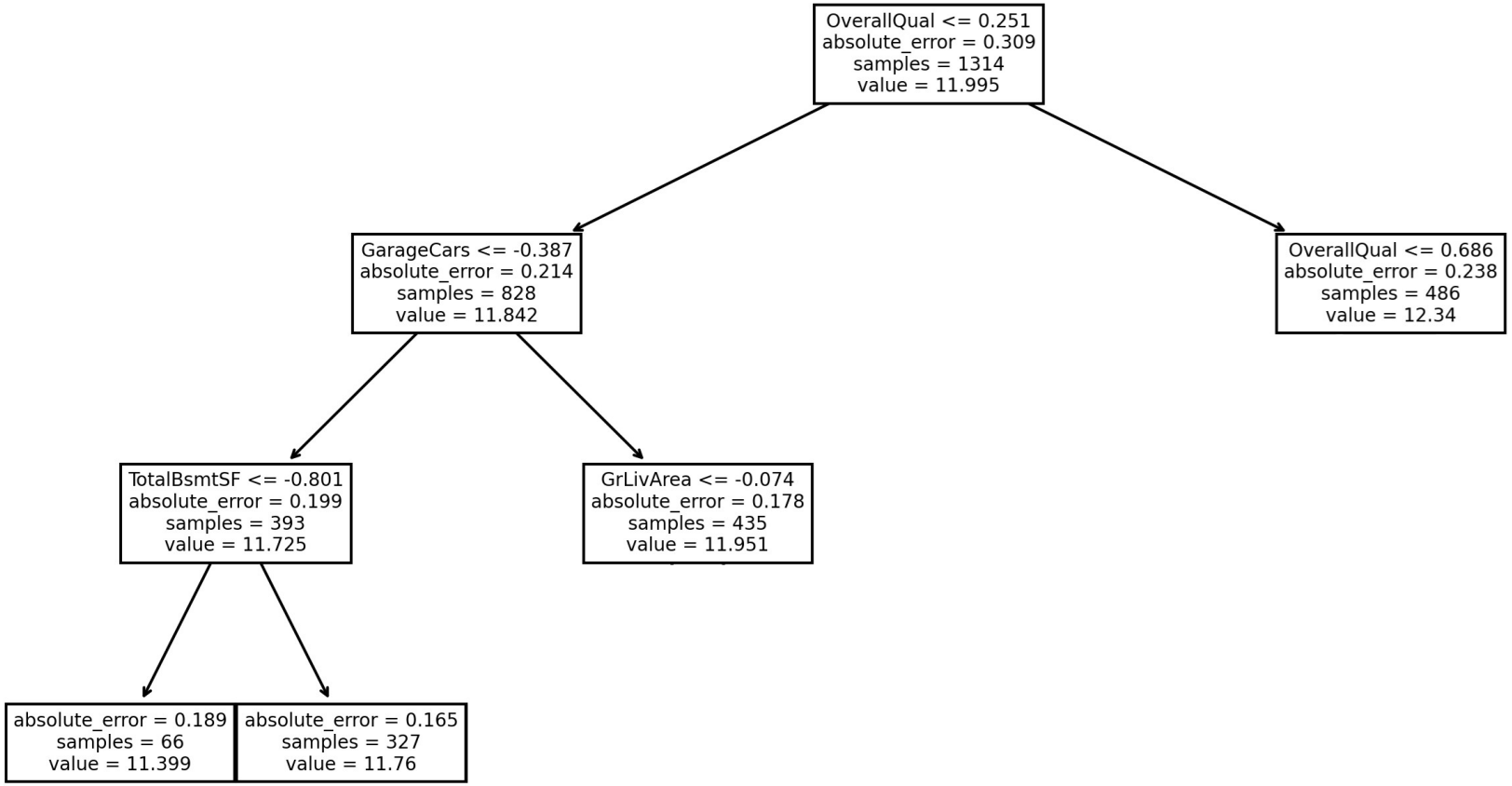
An optimization approach to learning falling rule lists, C. Chen and C. Rudin (2017)

Falling rule lists



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

Falling rule lists

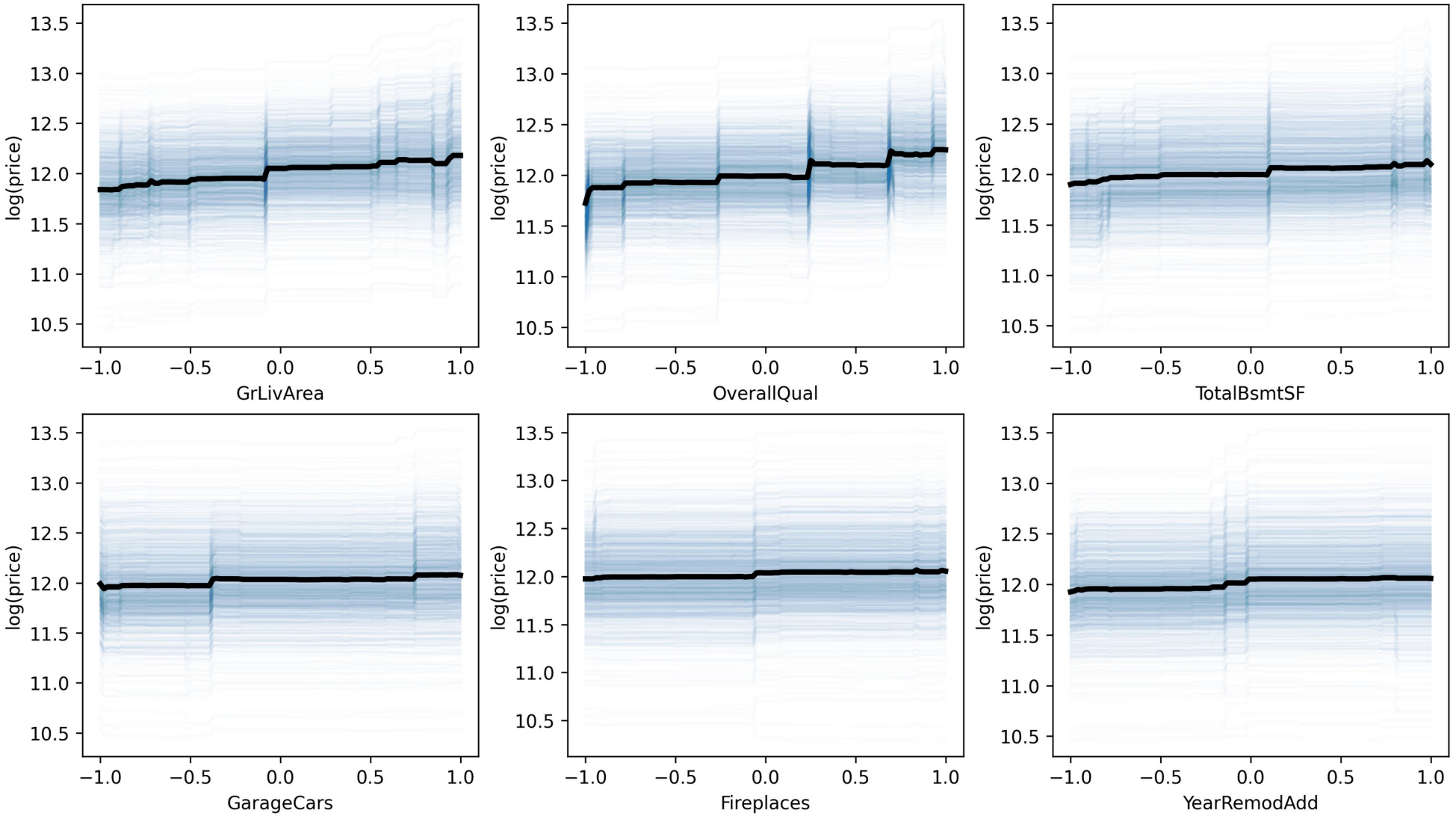


An optimization approach to learning falling rule lists, C. Chen and C. Rudin (2017)

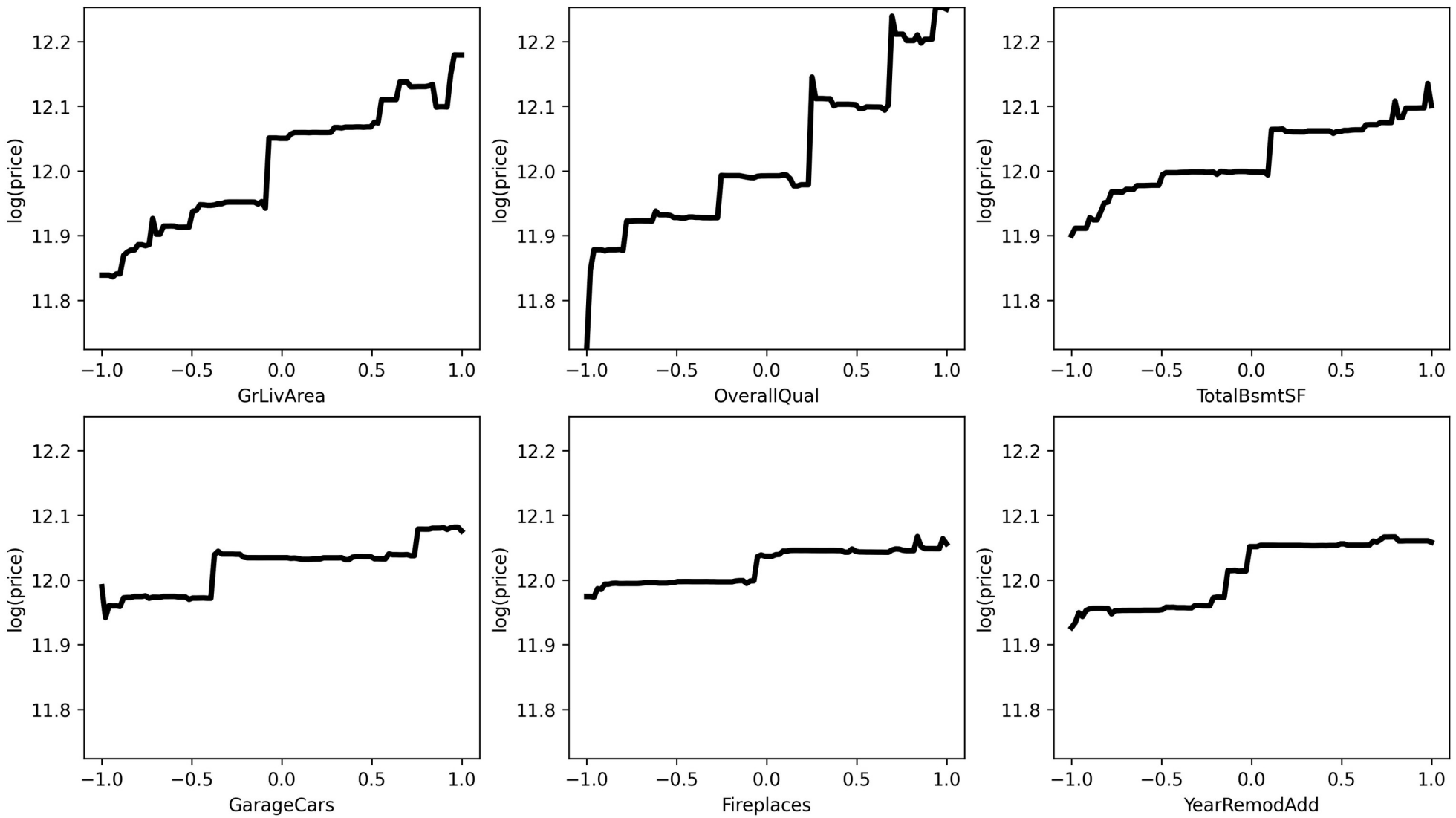
Boosting

- 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



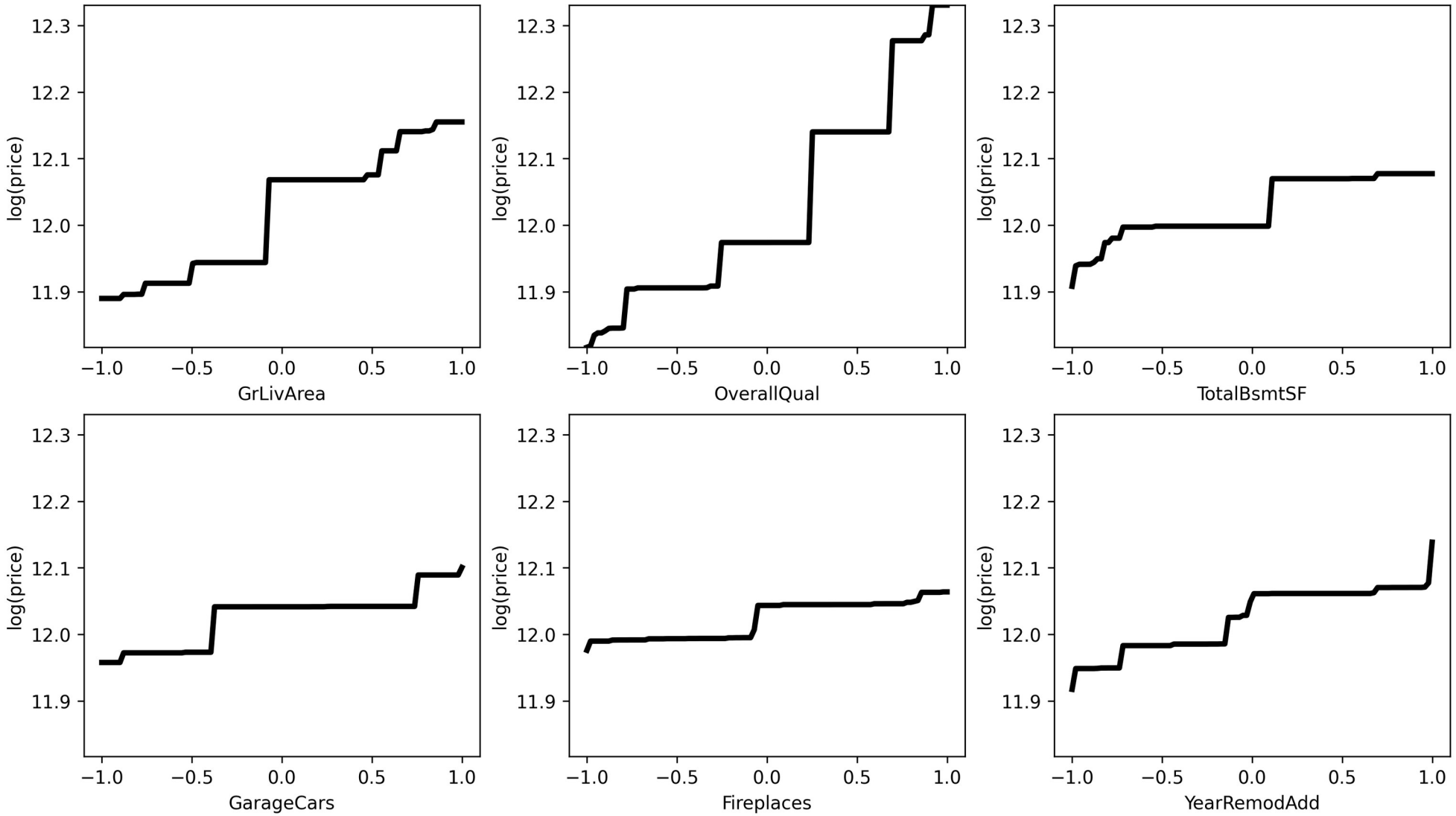
Gradient boosting



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)

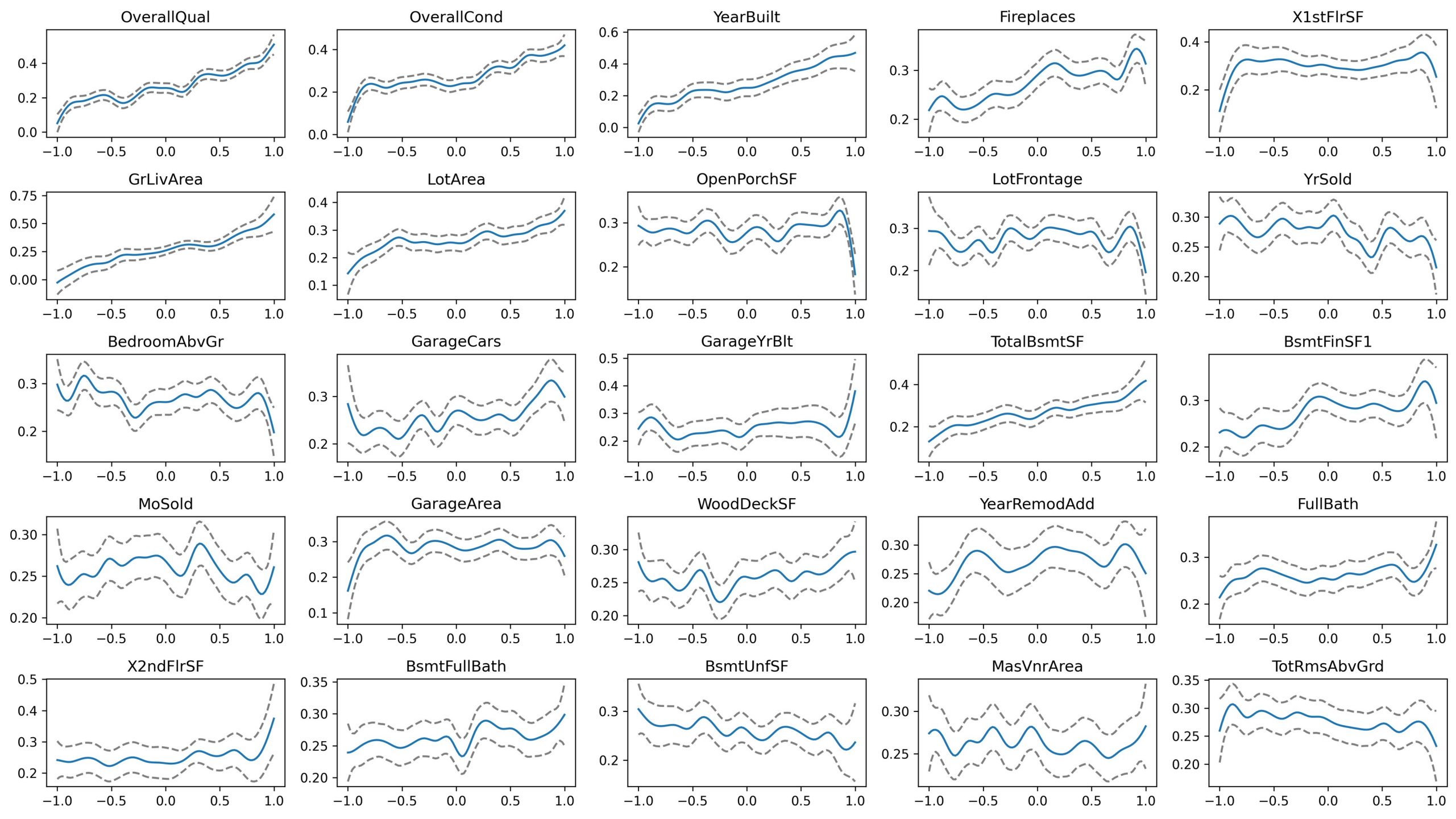
GAM

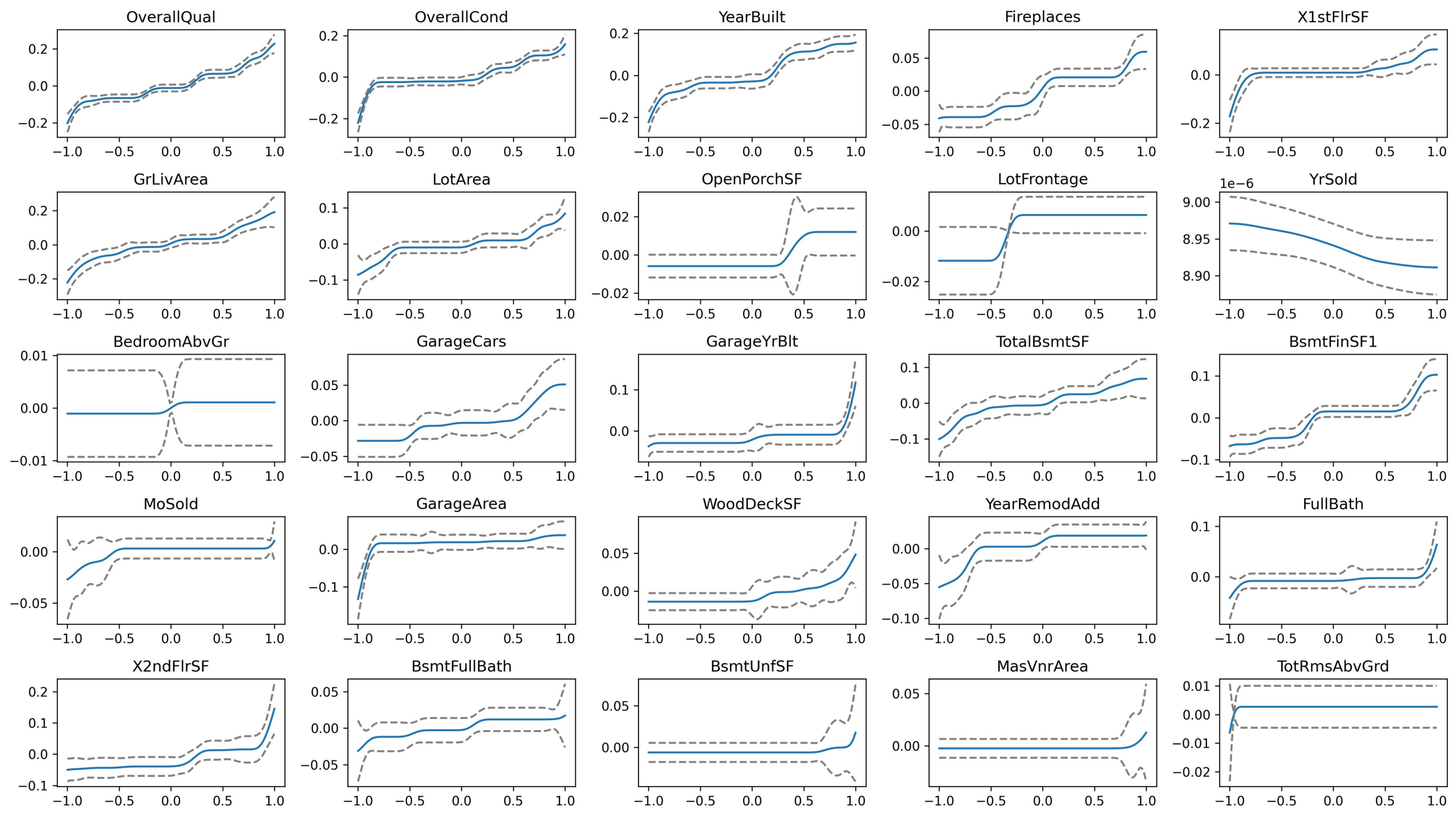
Find functions $f_1,\ldots,f_k$ such that

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

```
# 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)
```

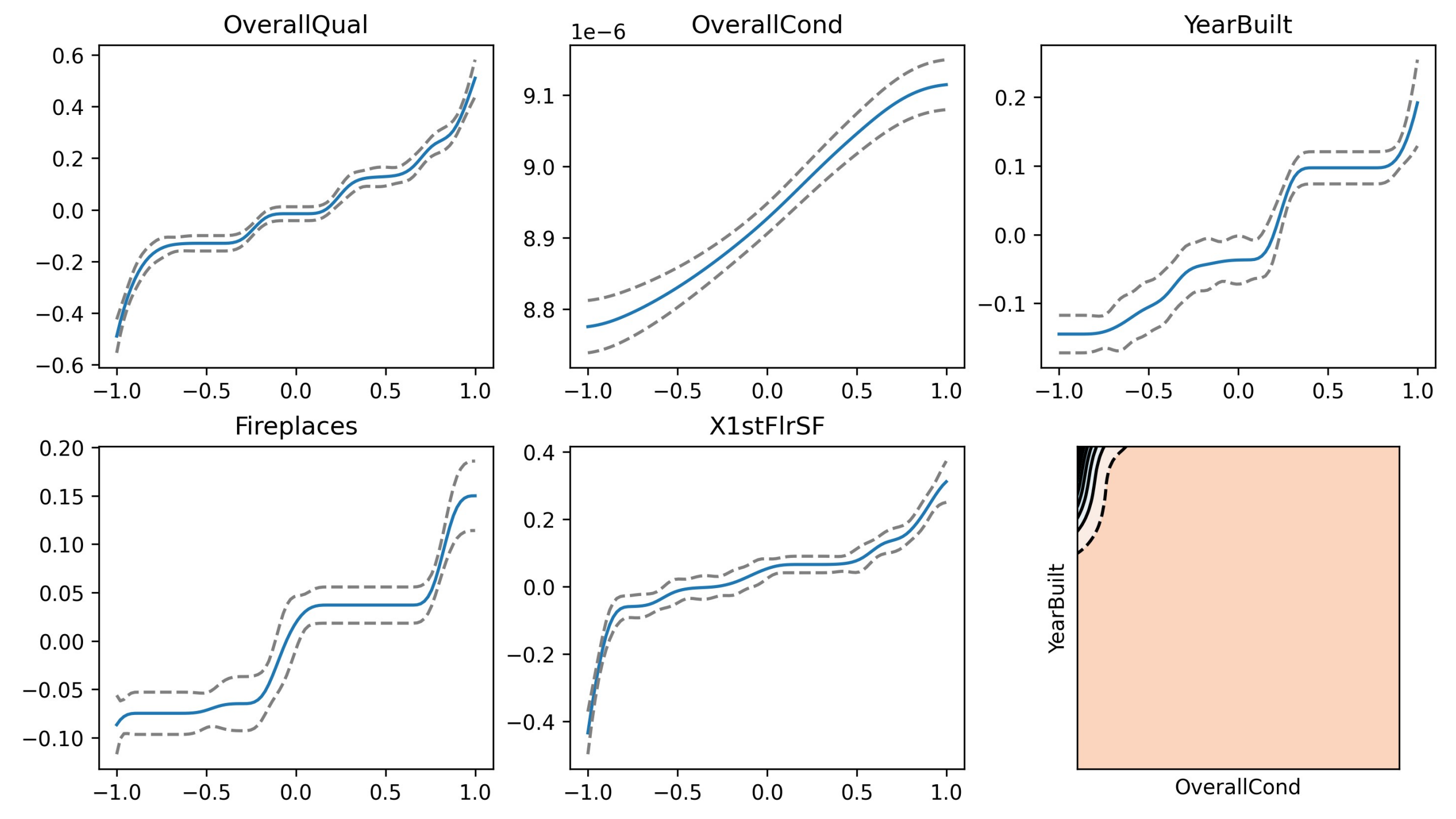
https://pygam.readthedocs.io/en/latest/


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

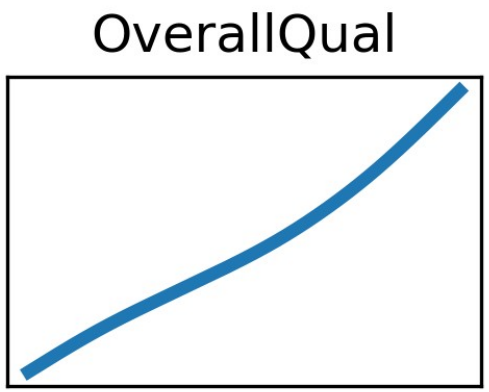
Boosting with GAMs as base learners:

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

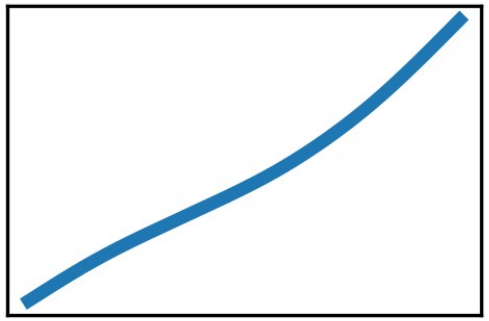
This gives a GAM *regularization path*.

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

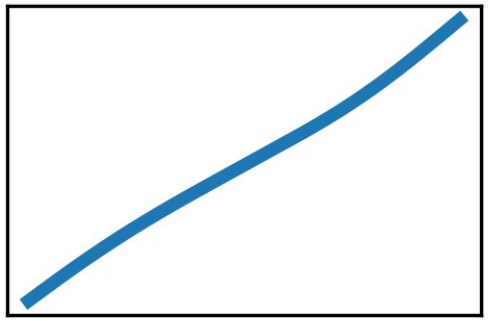
Model-based boosting in R: a hands-on tutorial using the R package mboost (B. Hofner et al., 2014)

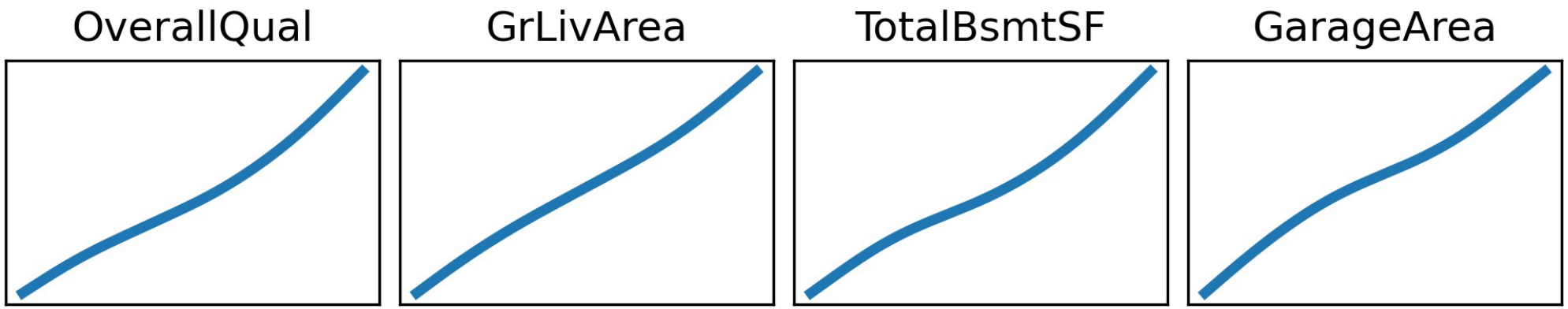

OverallQual

OverallQual

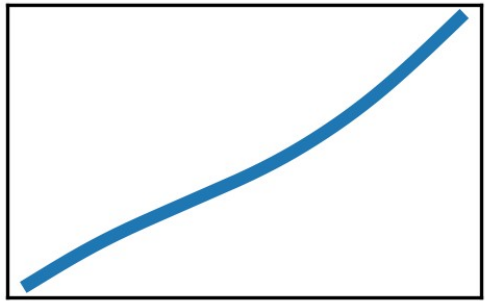


GrLivArea

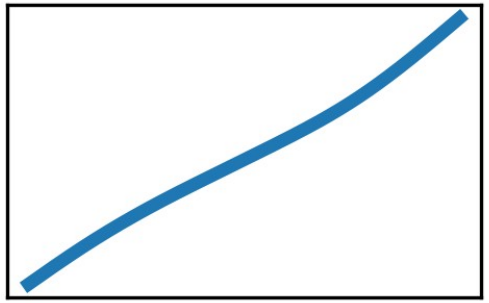



OverallQual
GrLivArea
TotalBsmtSF
GarageArea

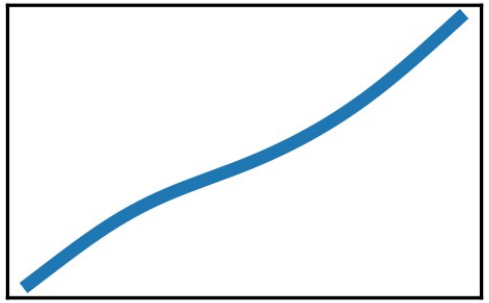
OverallQual



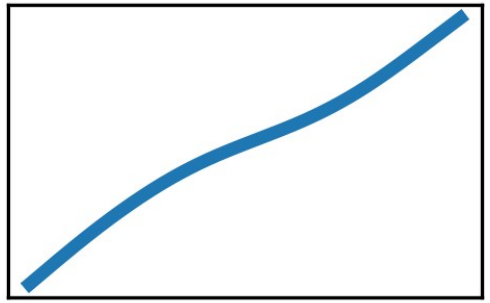
GrLivArea



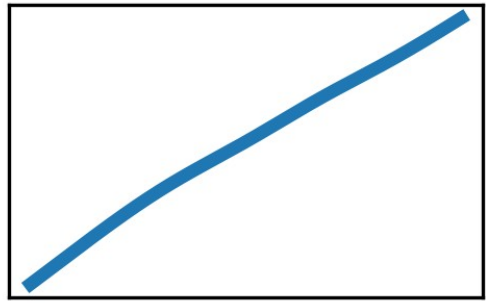
TotalBsmtSF



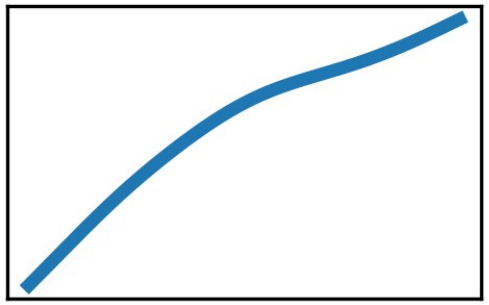
GarageArea



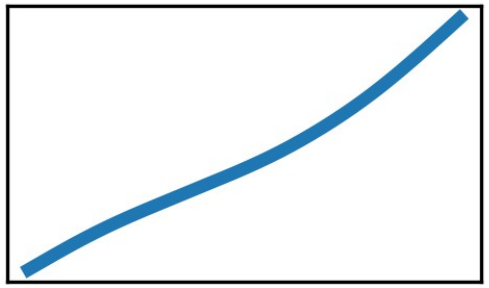
YearBuilt



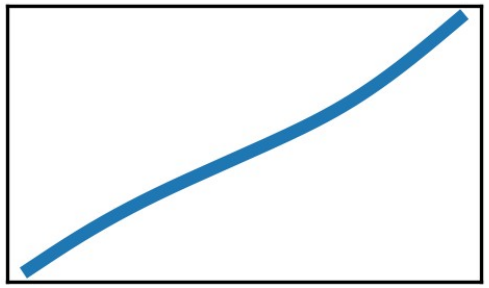
YearRemodAdd



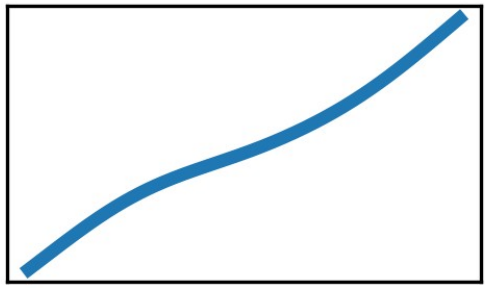
OverallQual



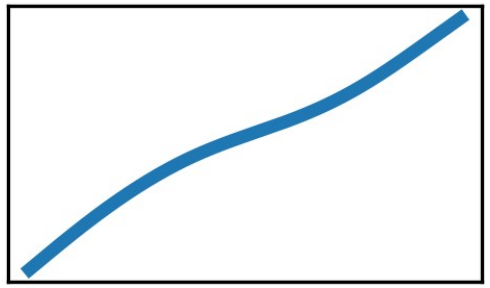
GrLivArea



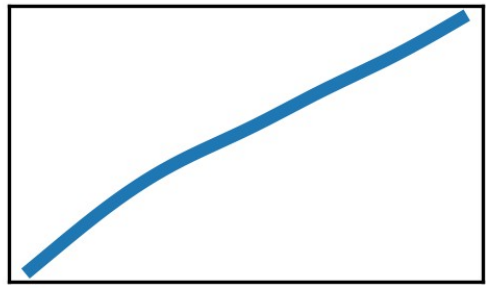
TotalBsmtSF



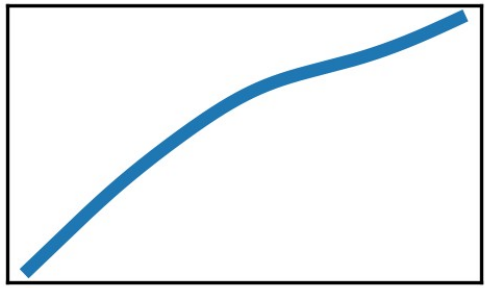
GarageArea



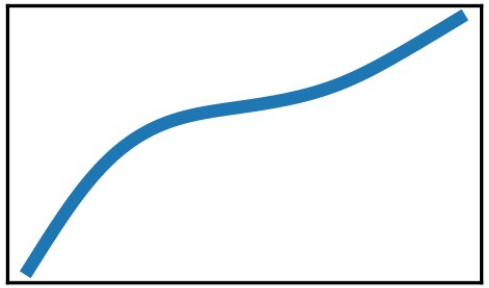
YearBuilt



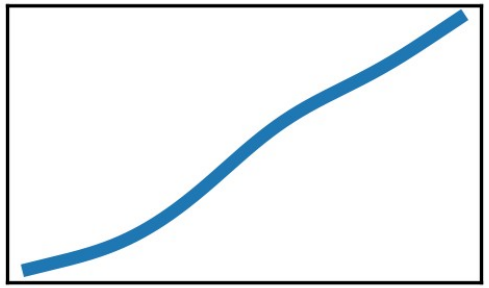
YearRemodAdd



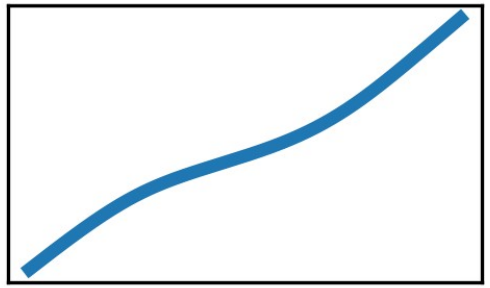
OverallCond



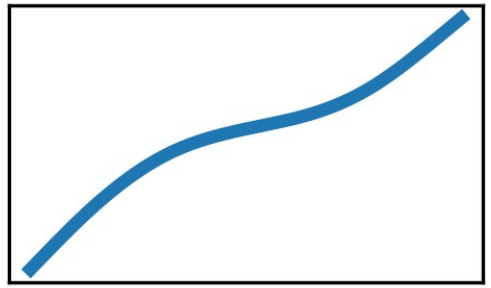
Fireplaces



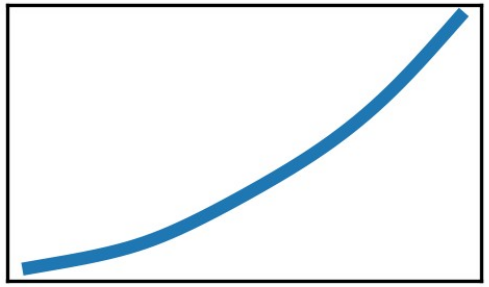
LotArea

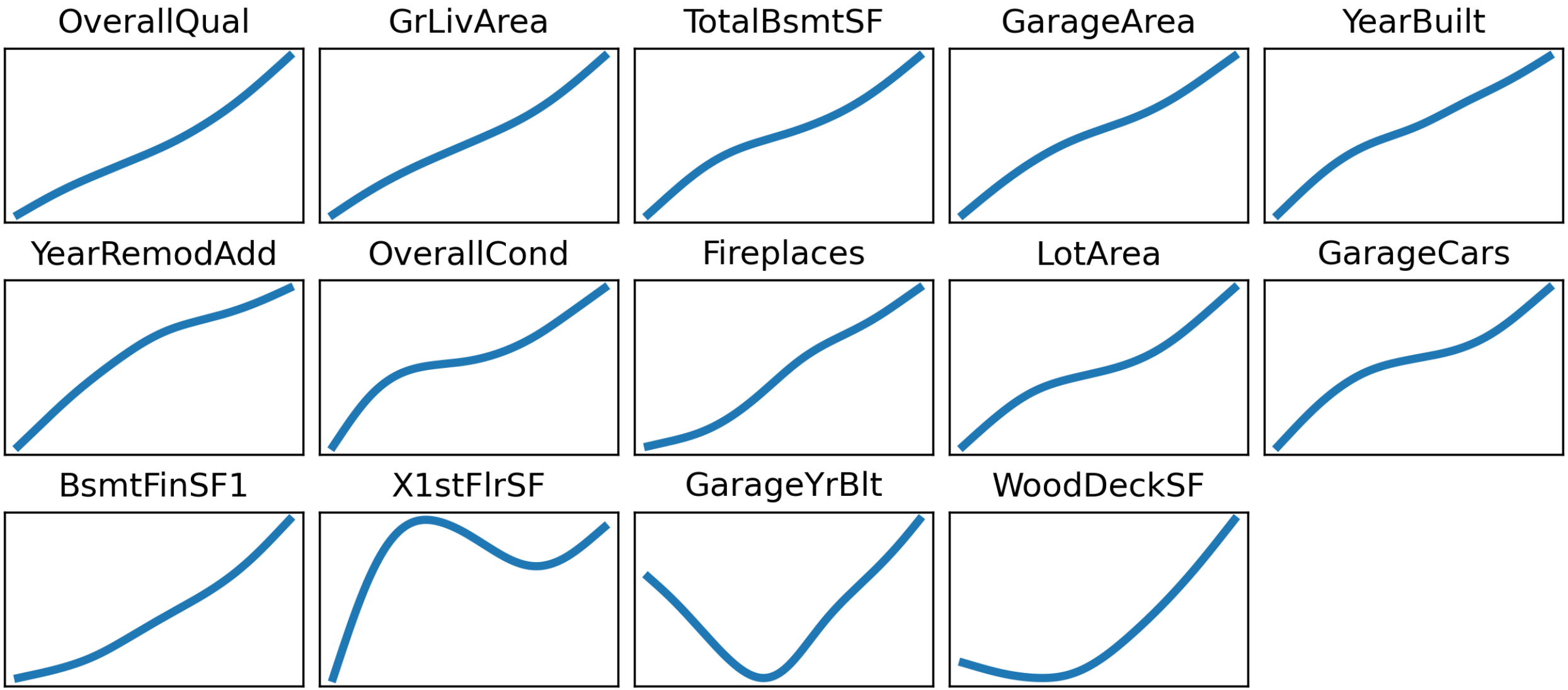


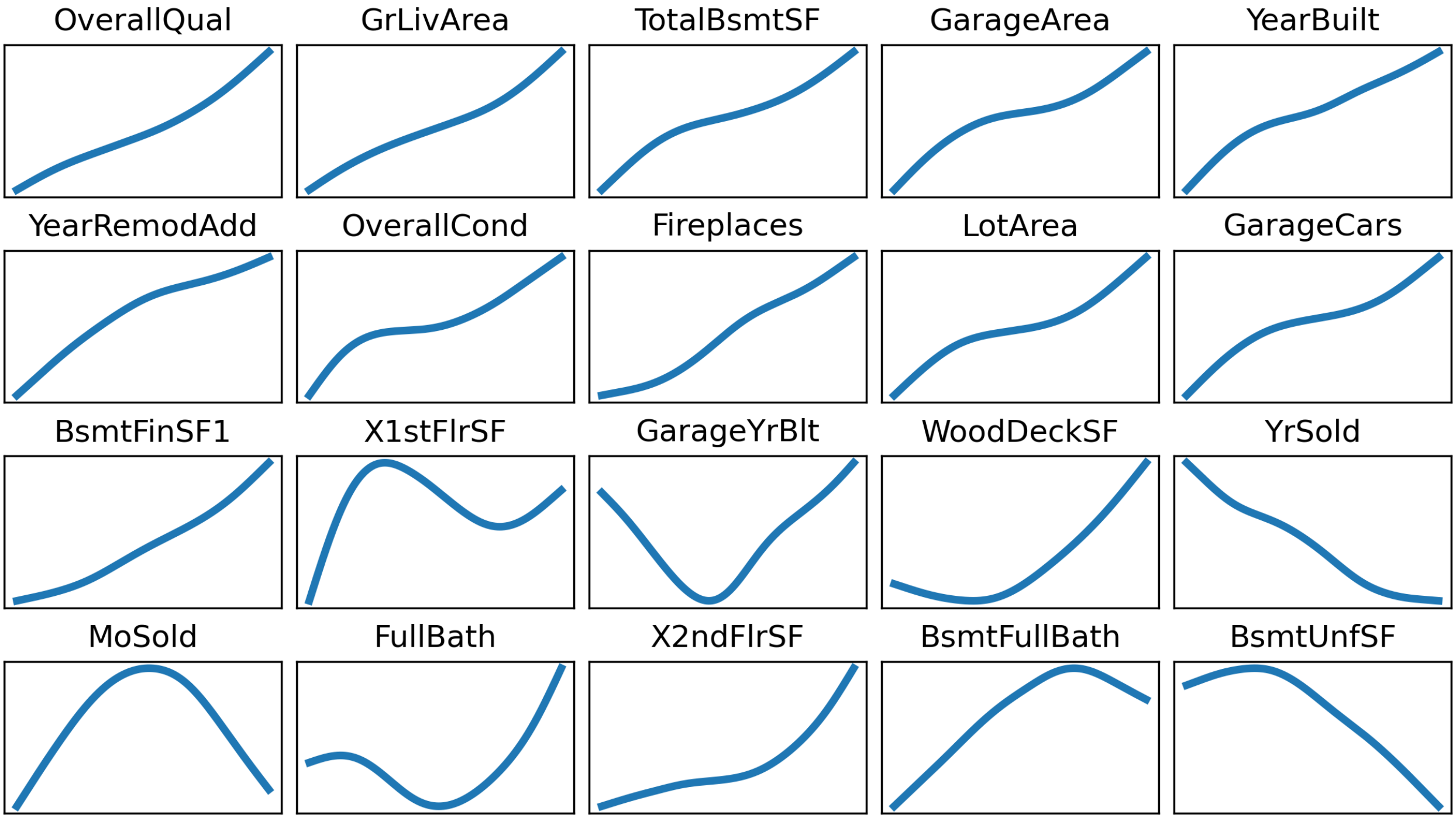
GarageCars

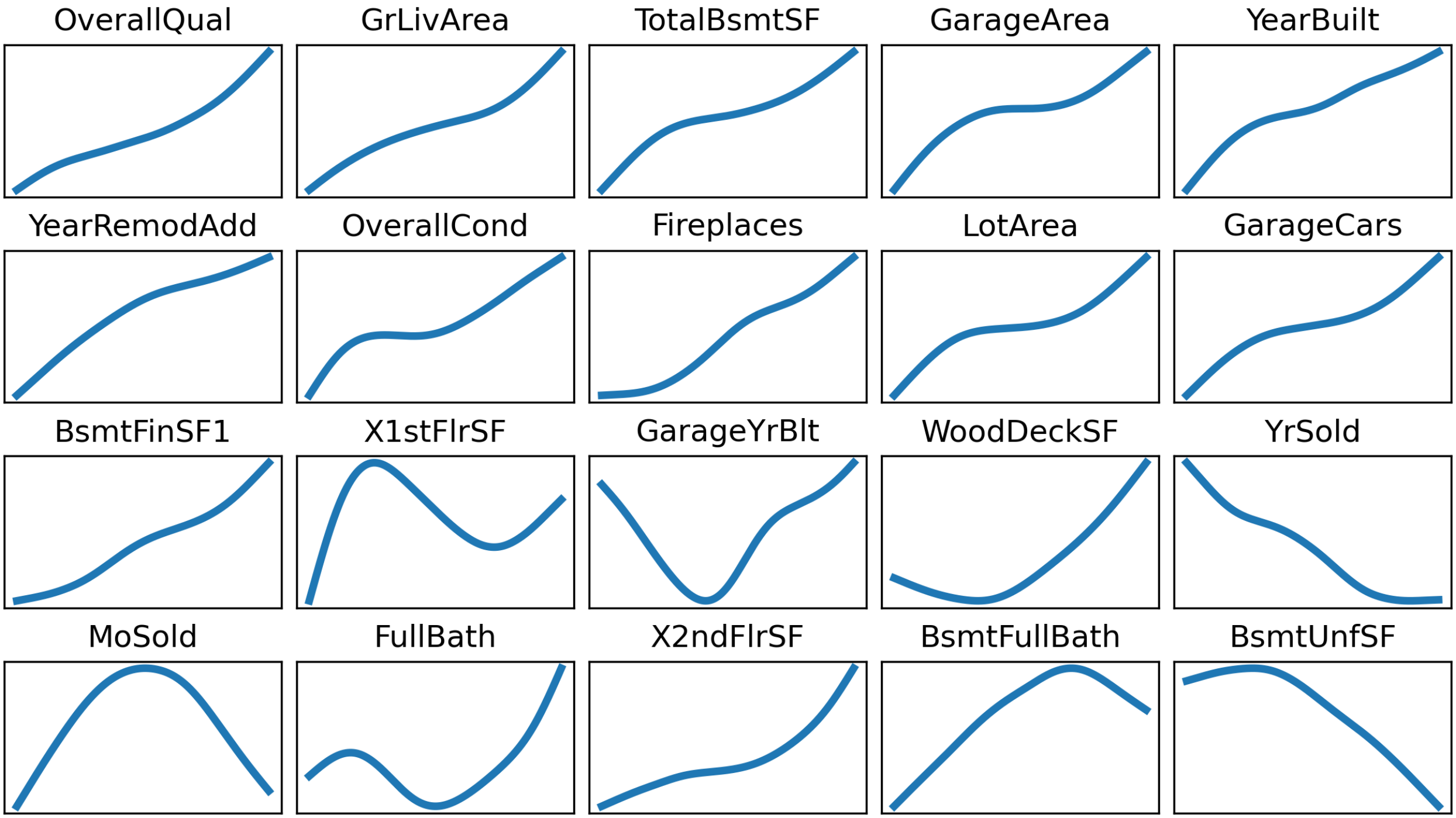


BsmtFinSF1




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:

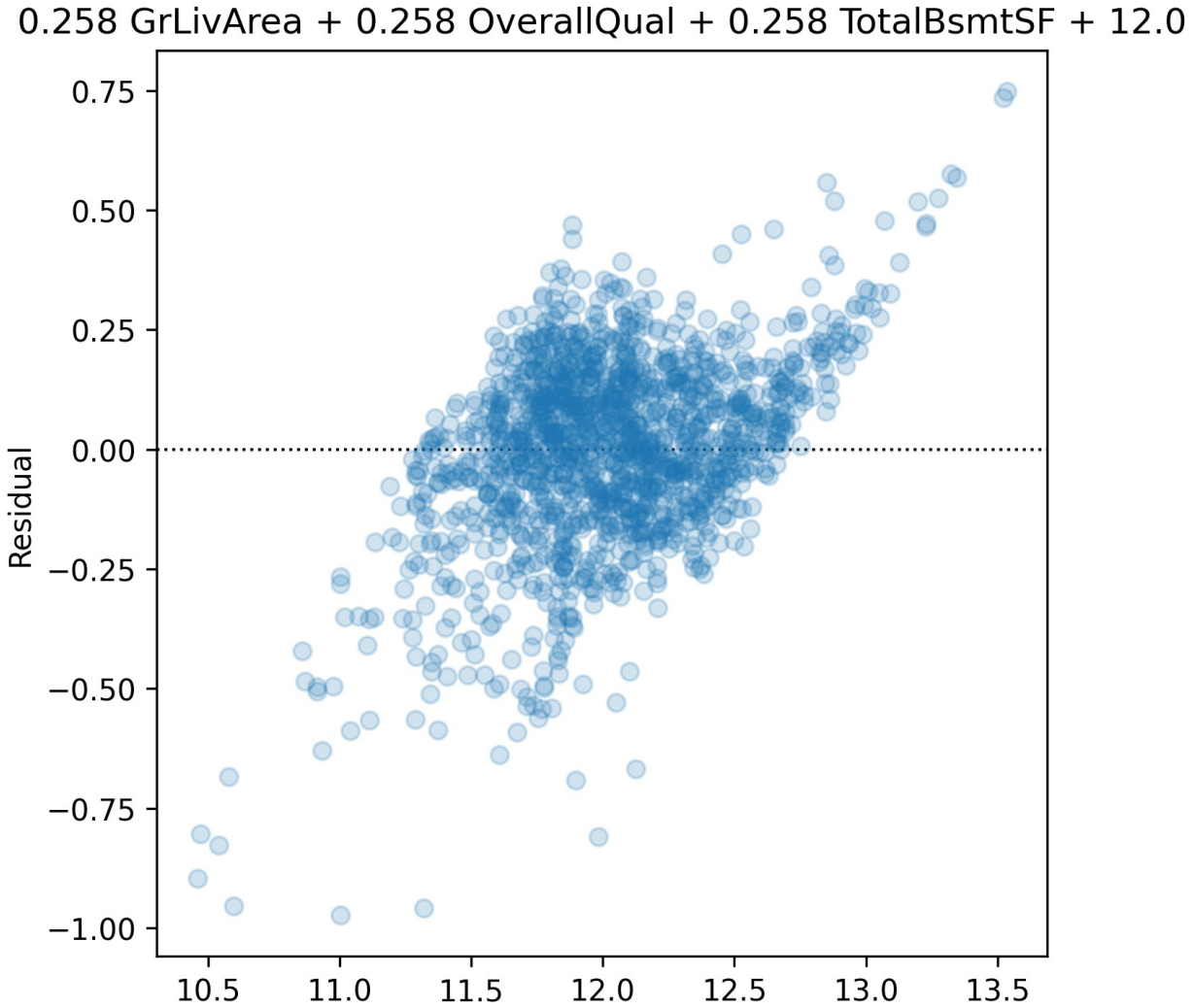
- Add an orthogonality penalty on β, $\|B'B-I\|^2$
- Add a sparsity penalty, $\|B\|_1$

```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)

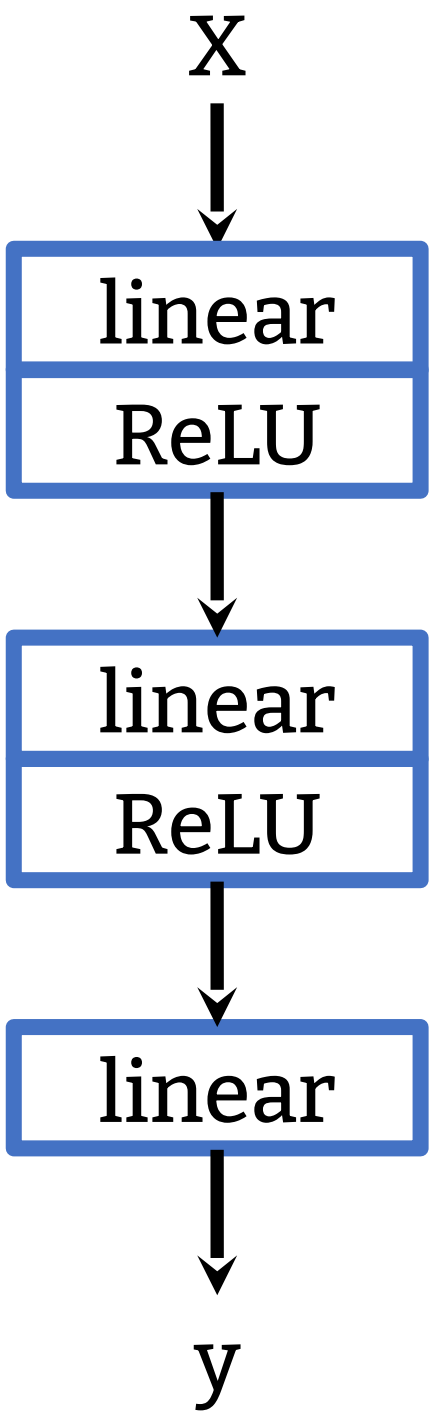
Deep Learning

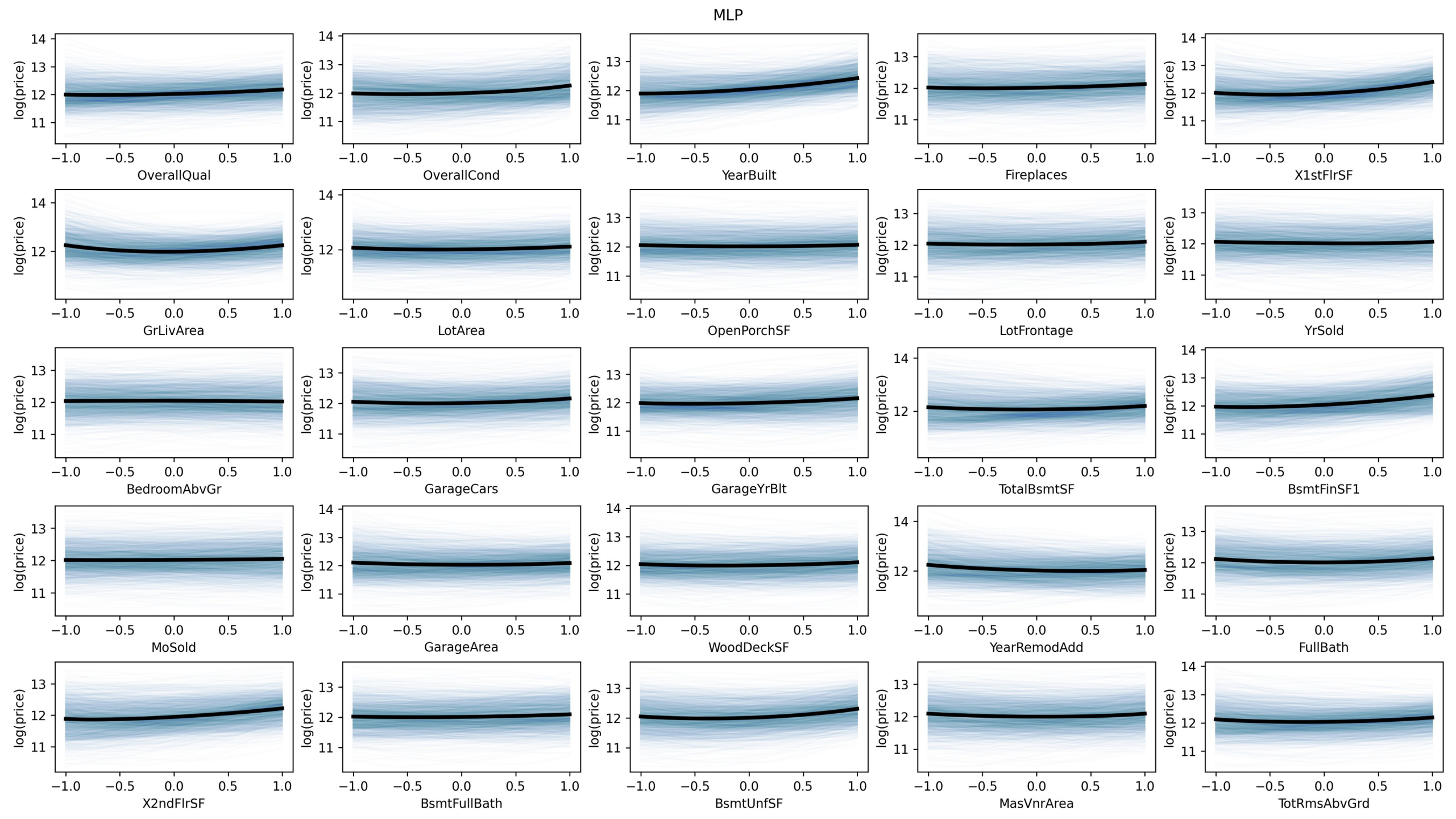
Deep learning

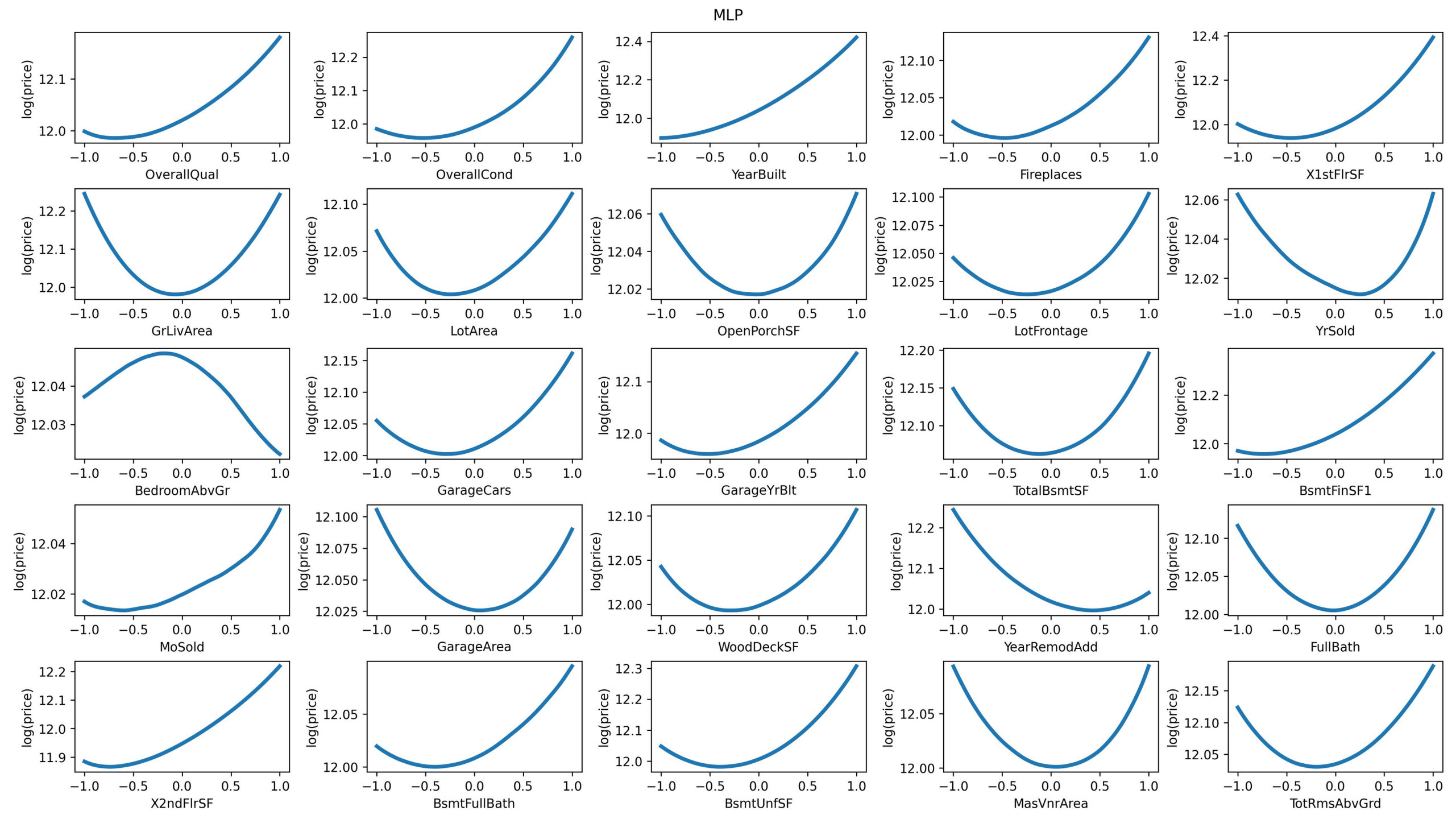
- Sparse-input neural nets
- GAMI-Net
- Monotonic MLP: gradient penalty, MIP monotonicity proof, adversarial training
- Varying coefficient models, self-explaining neural nets
- Mixtures of experts

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

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

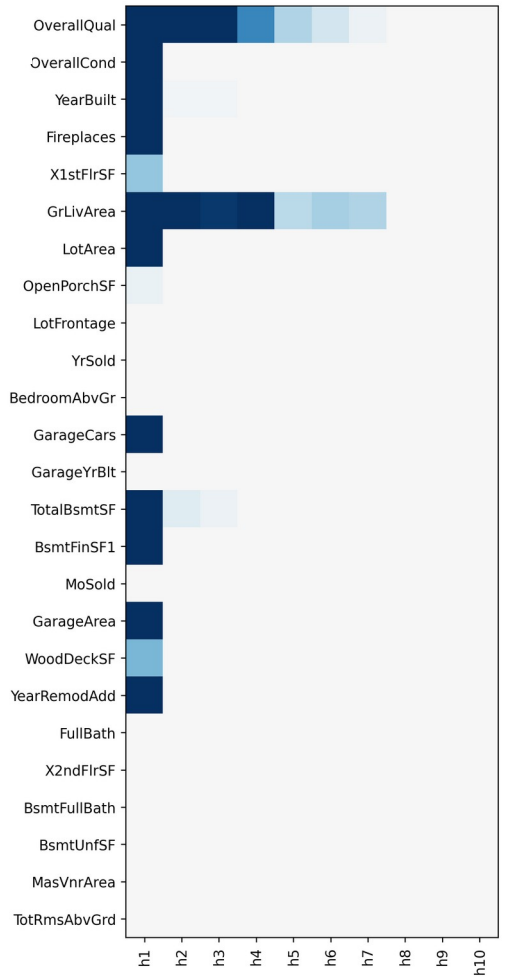
$$w_1 \leftarrow S(w_1, \gamma\lambda_1) \qquad \text{(soft-thresholding)}$$

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

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

Sparse-input neural nets

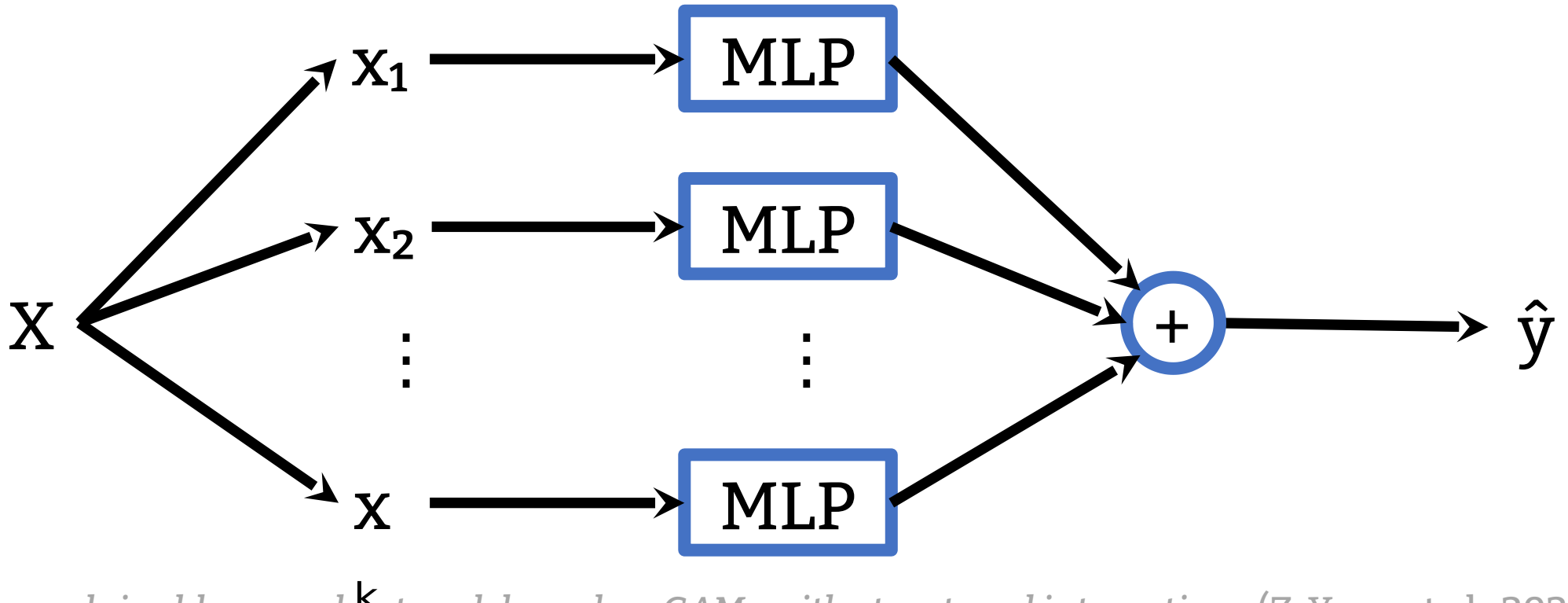
```
error = loss_fn( y_hat, y_ )
penalty_12 = torch.linalg.vector_norm( model.fc1.weight, ord=2, dim=0 ).sum()
penalty_1  = torch.linalg.vector_norm( model.fc1.weight, ord=1, dim=0 ).sum()
penalty_2  = (
    torch.linalg.vector_norm( model.fc2.weight, ord=2, dim=None ) ** 2 +
    torch.linalg.vector_norm(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

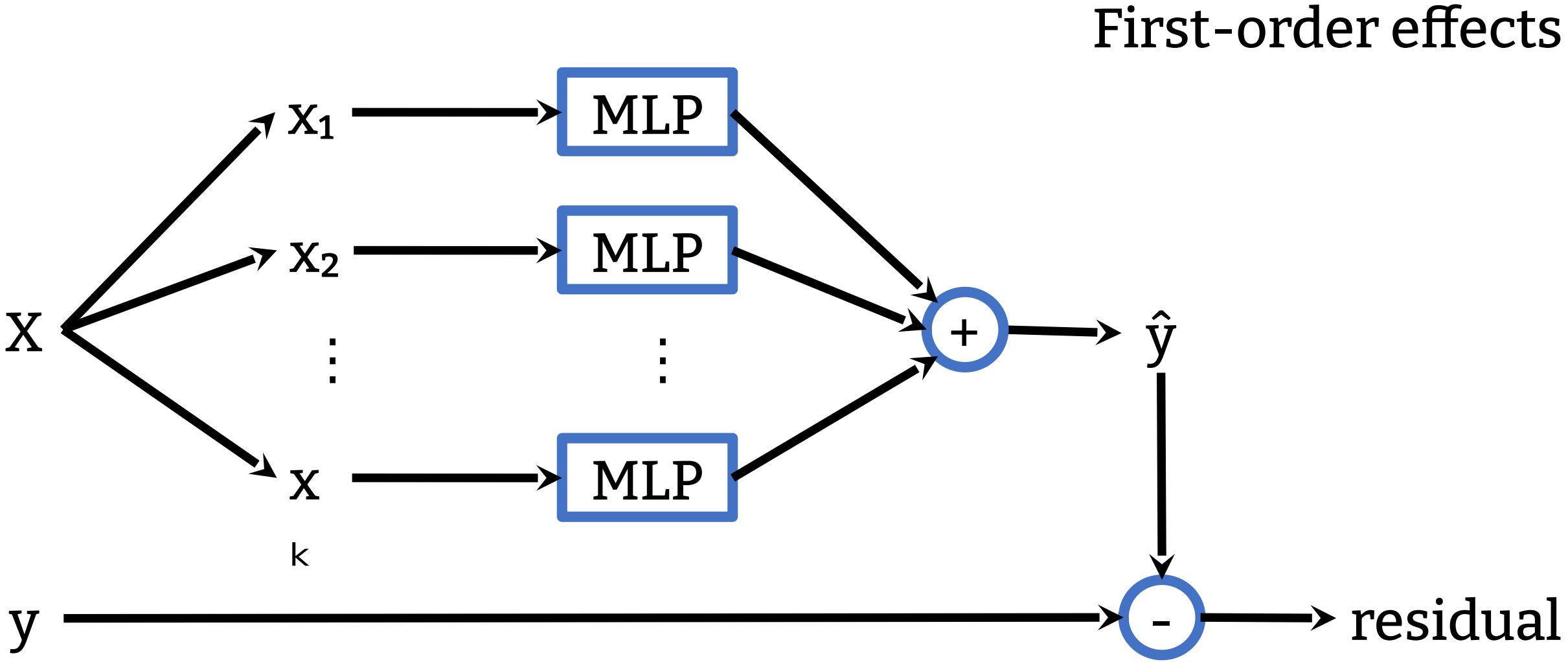
GAMI-Net

- 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)

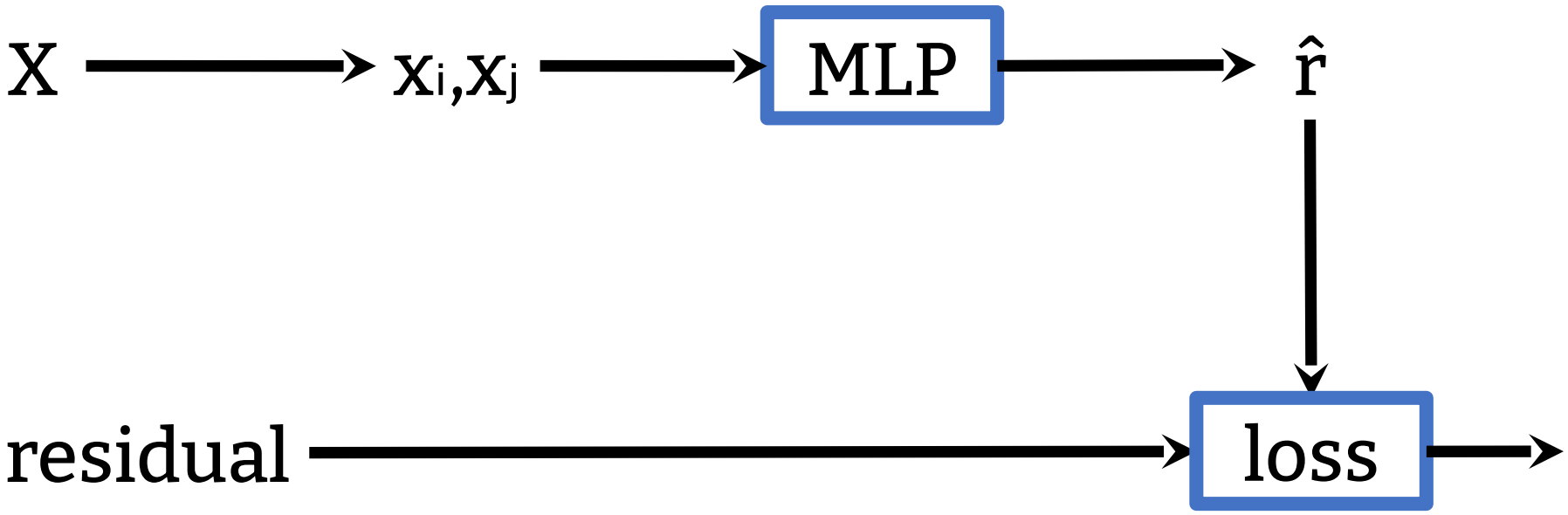
GAMI-Net



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

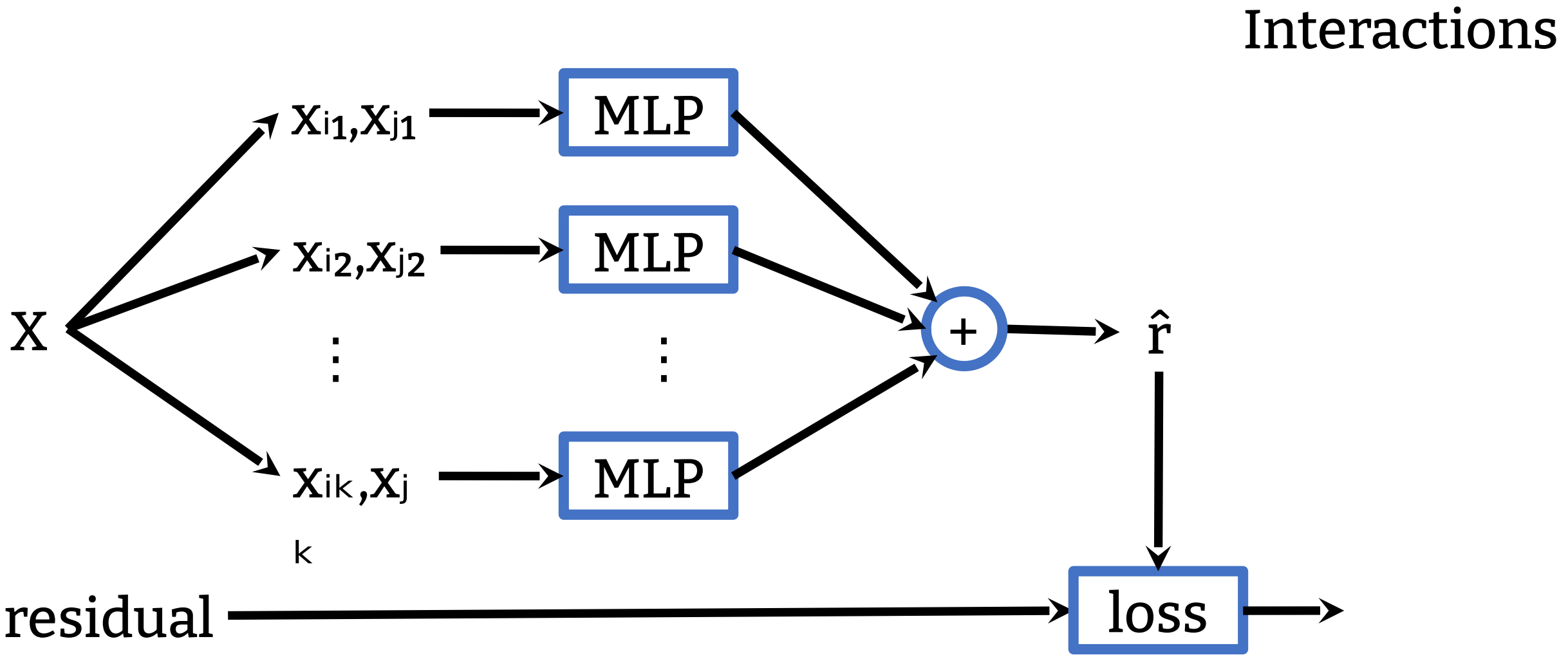
GAMI-Net

Interaction selection



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

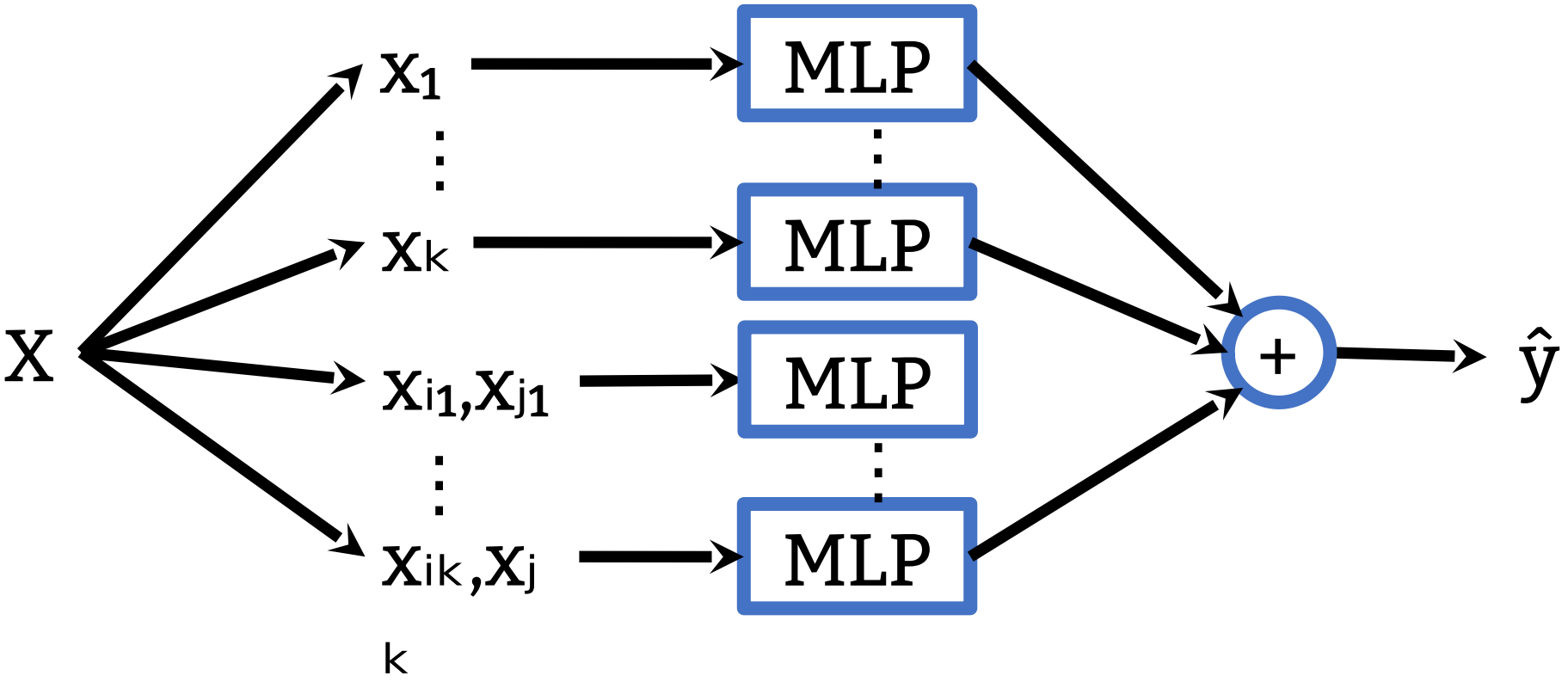
GAMI-Net



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

GAMI-Net

Fine-tuning



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

GAMI-Net: penalties

- Sparsity penalty
- Only include interactions if the corresponding main effects have been selected
- Orthogonality between main effects and interactions

$$\left\langle f_{ij}(x_i, x_j), f_i(x_i) \right\rangle = 0$$

$$\left\langle f_{ij}(x_i, x_j), f_j(x_j) \right\rangle = 0$$

GAMI-Net

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

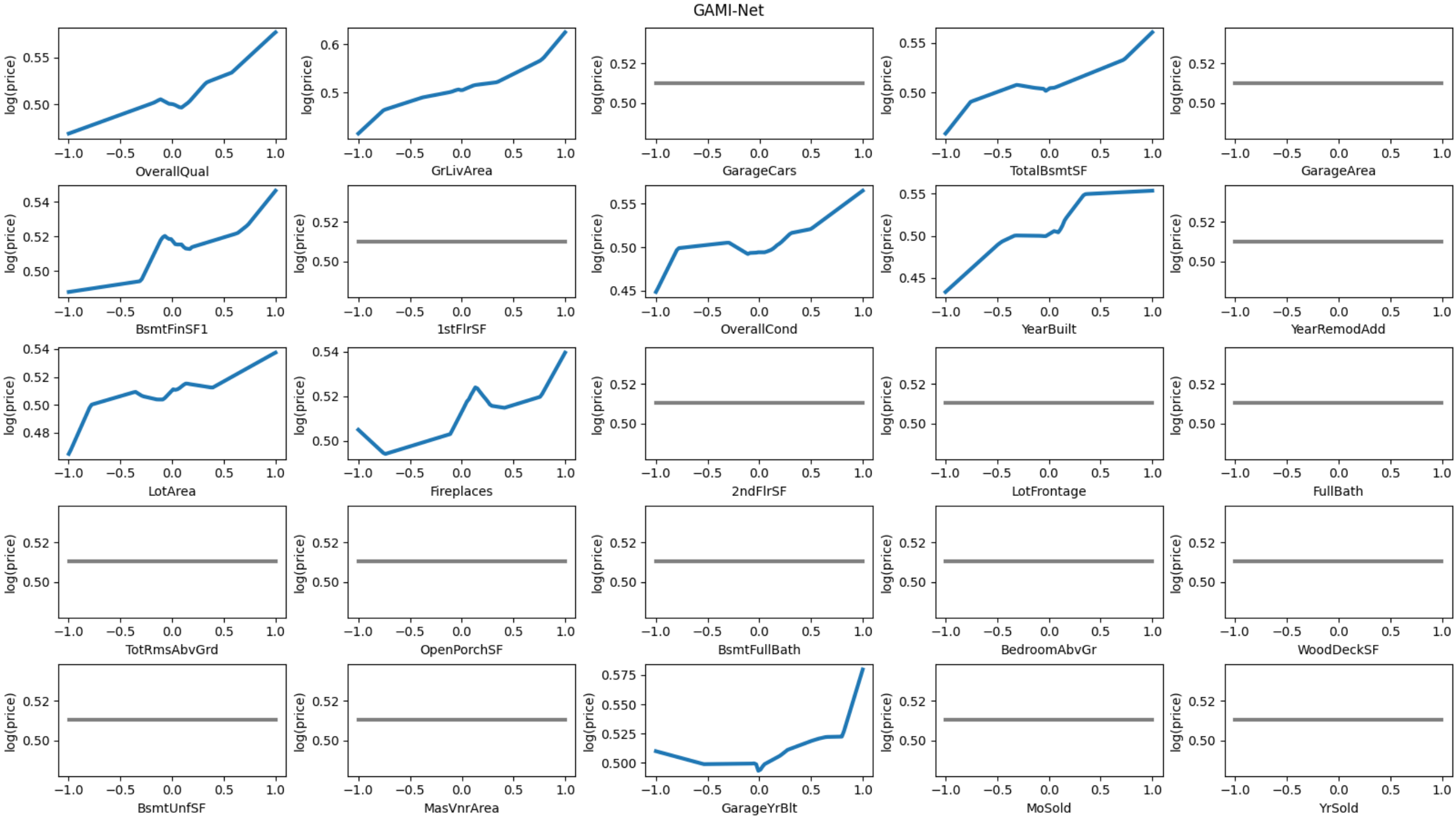
```
from piml.models import GAMINetRegressor
from piml import Experiment

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")
```

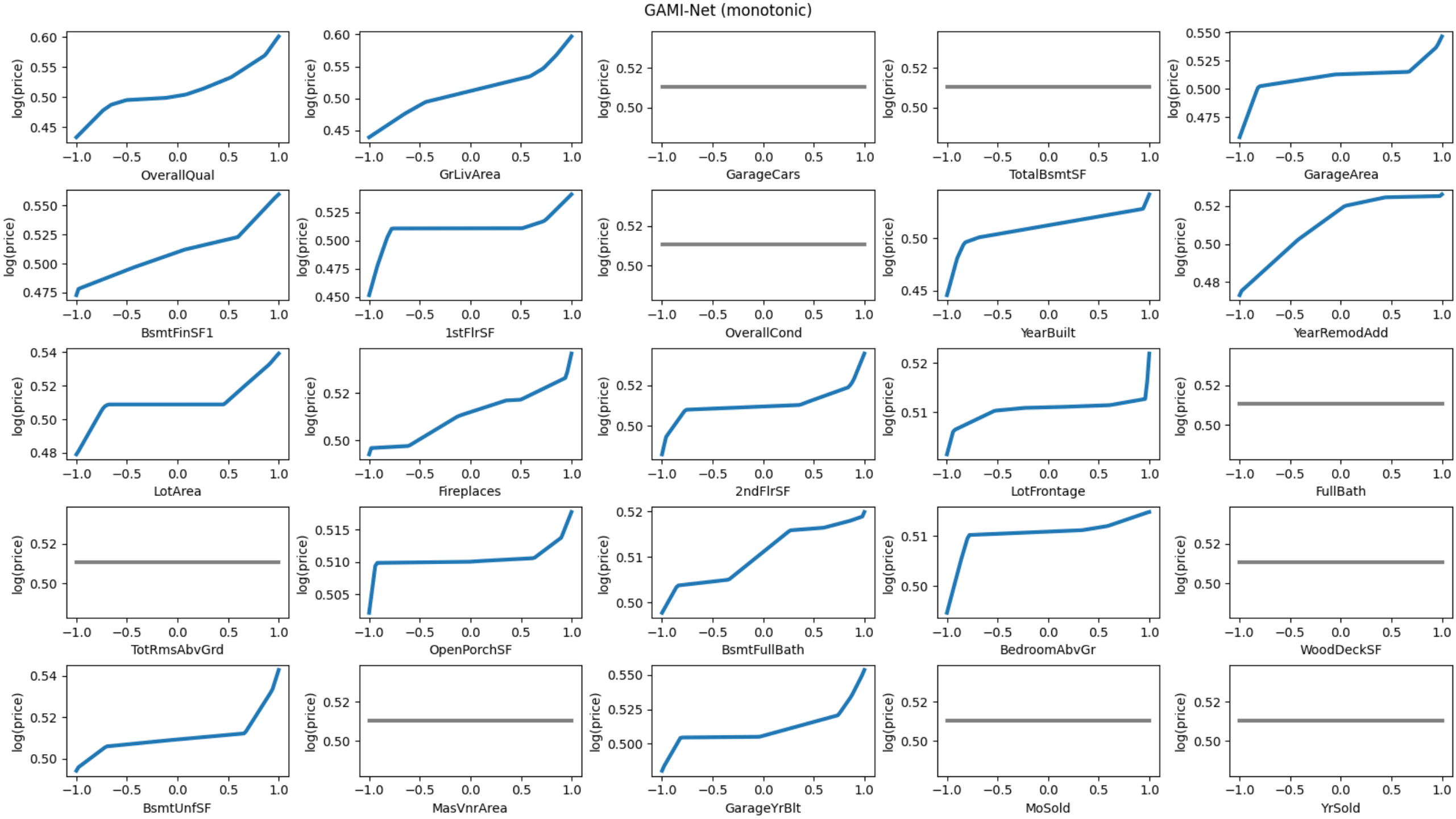
GAMI-Net: an explainable neural network based on GAMs with structured interactions (Z. Yang et al., 2020)

GAMI-Net

- Fit the main effects; set their means to zero; prune them (only keep those with the largest variance)
- Fit interaction effects on the residuals, one pair of variables at a time, and only keep the pairs with the largest variance
- Fit the interaction effects on the residuals, all at once, with a penalty to make them orthogonal to the main effects; prune them
- Fine-tune the whole model (main and interaction effects) end-to-end

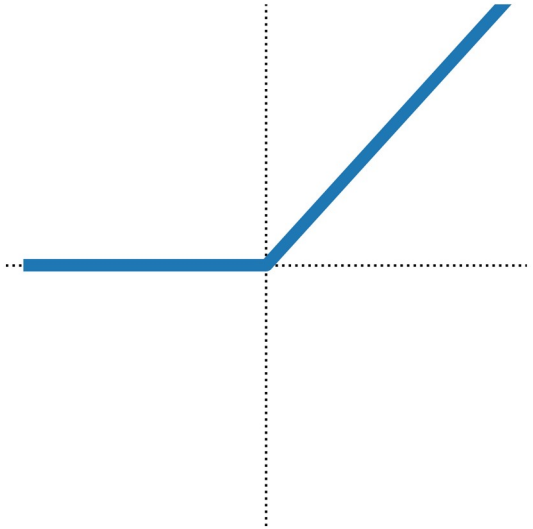
GAMI-Net: an explainable neural network based on GAMs with structured interactions (Z. Yang et al., 2020)


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

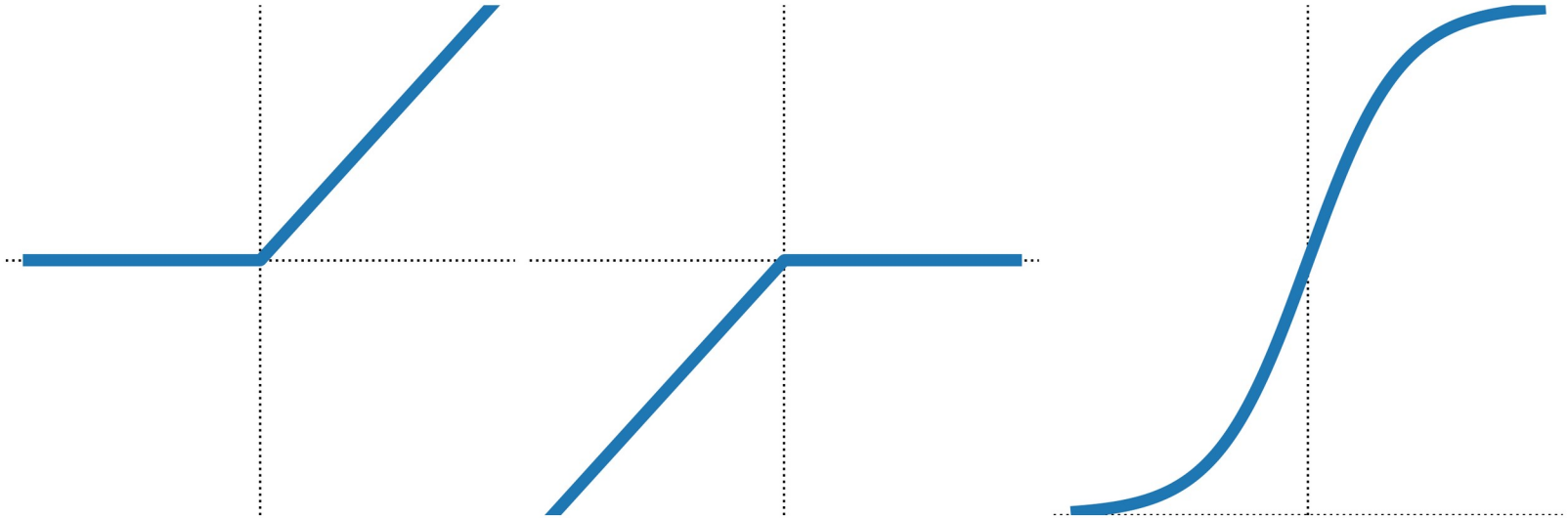
Monotonic MLP

- MLP with non-negative weights and ReLU activations: monotonic but convex



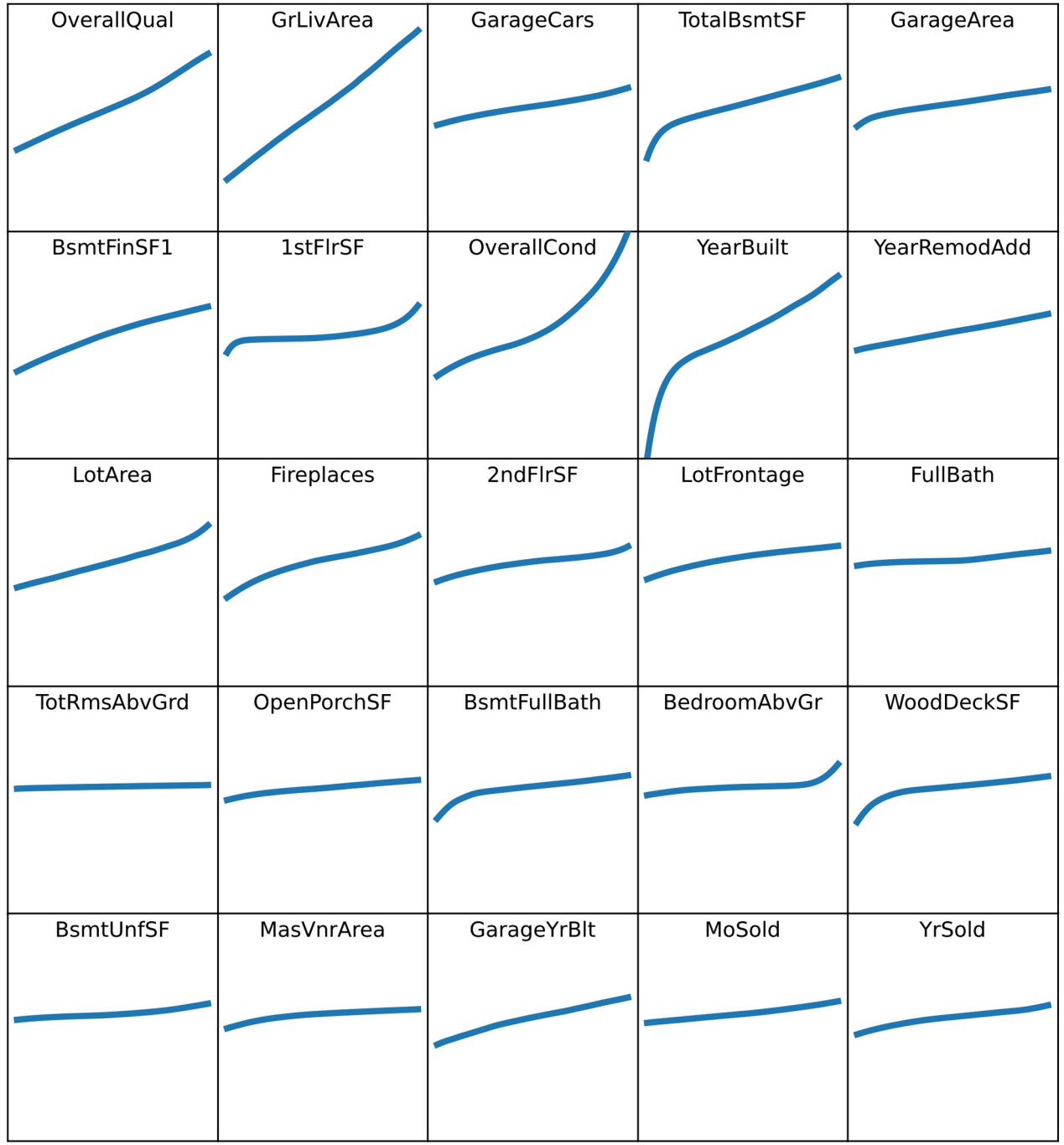
Monotonic MLP

- 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

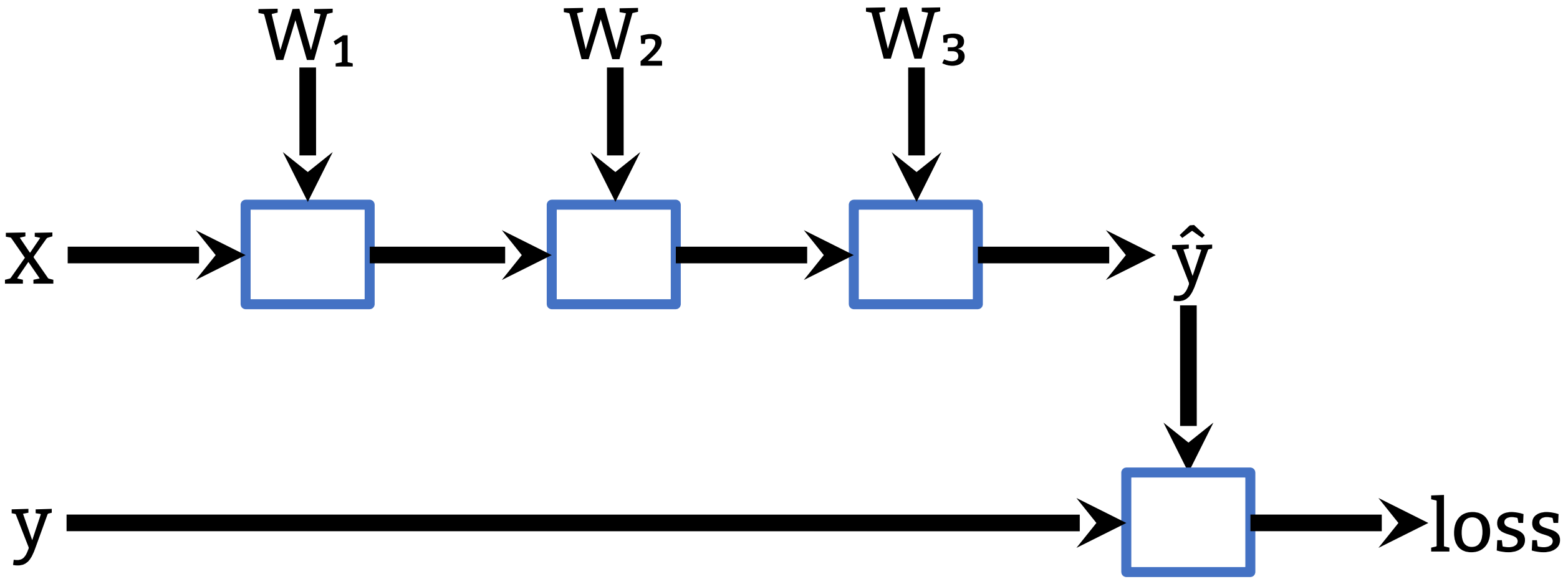
Add a penalty when the gradient ∂ŷ/∂X is negative.

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)

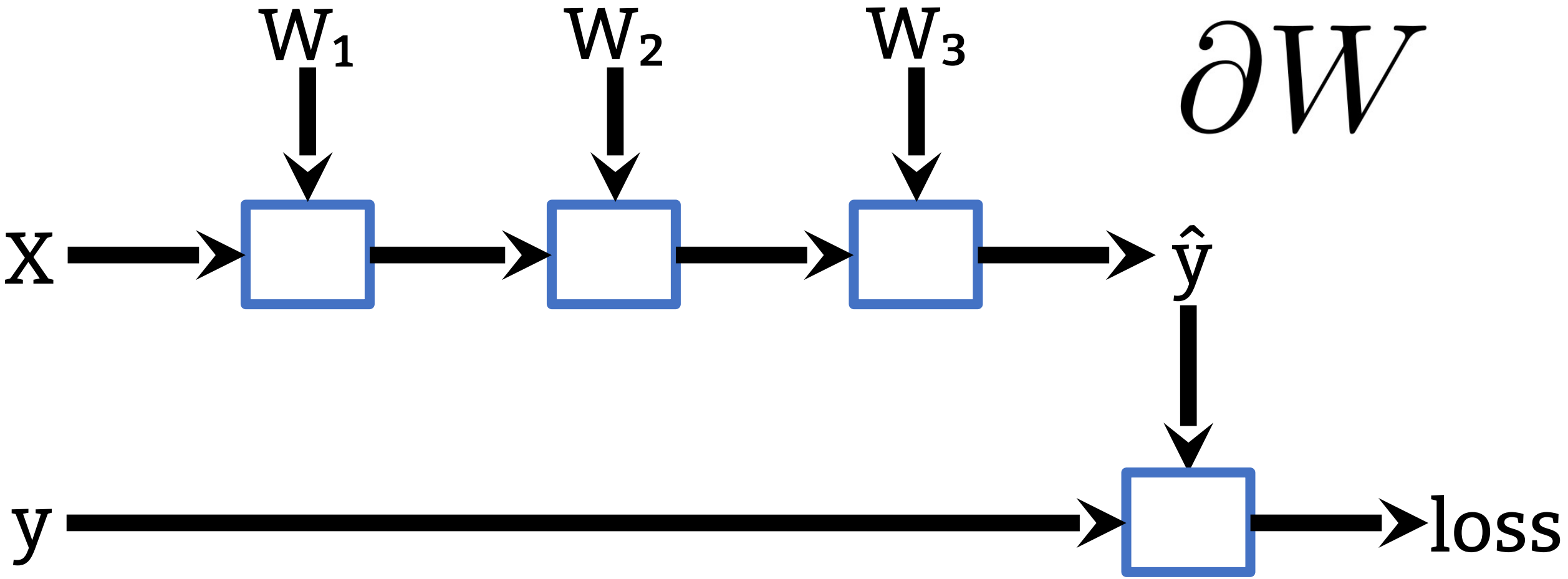
Gradient penalty



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

Gradient penalty

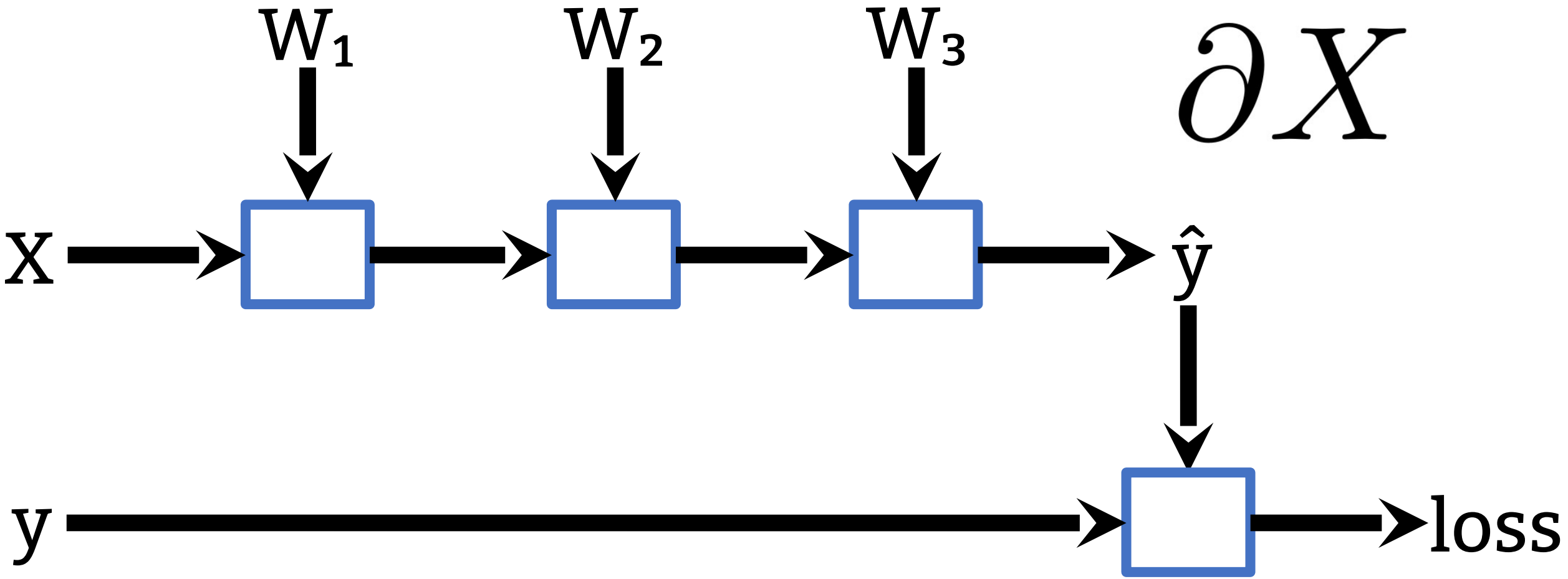
$$\frac{\partial \text{loss}}{\partial W}$$



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

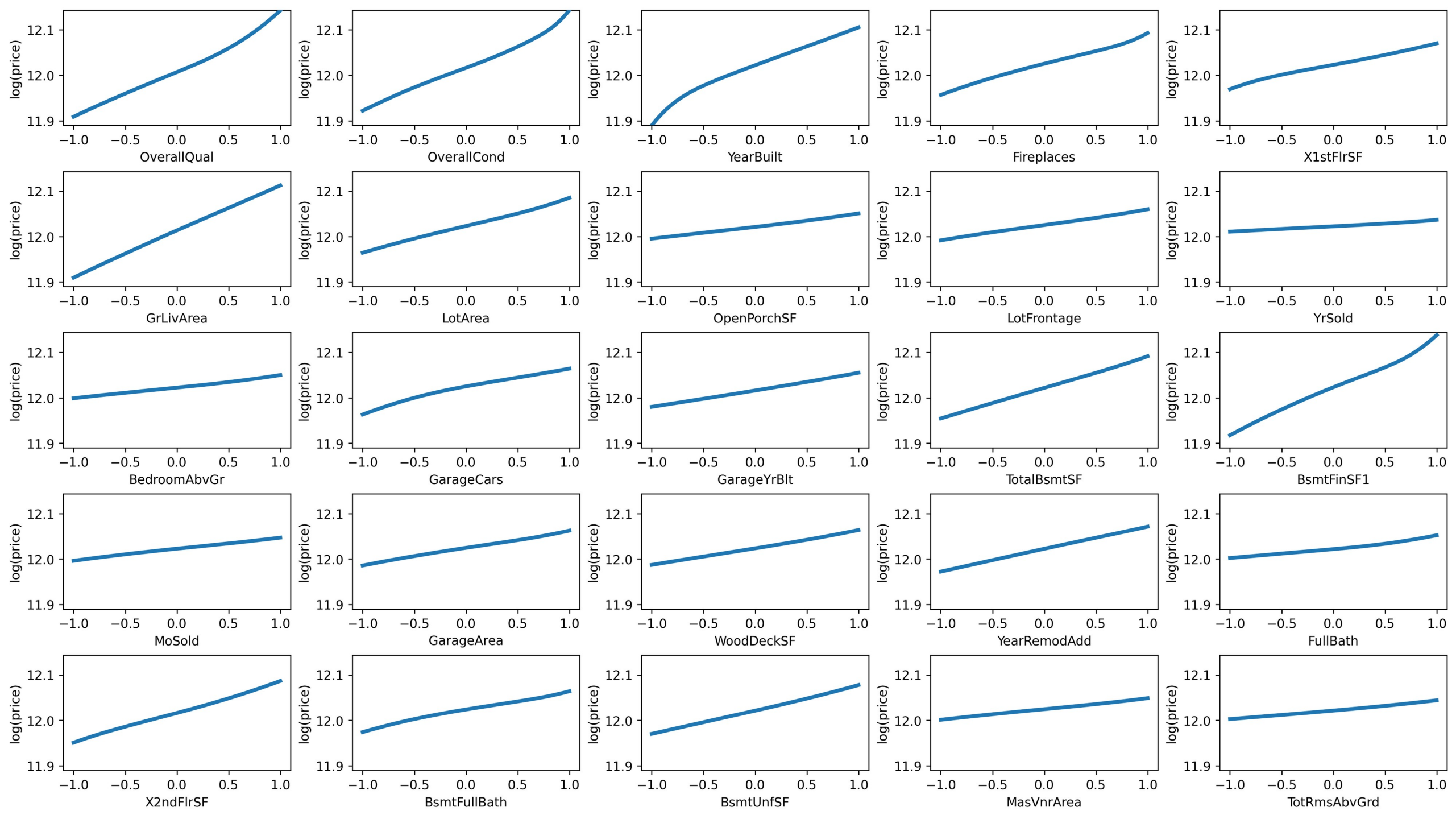
```
# Gradient penalty
for epoch in range(10_000):
    optimizer.zero_grad()
    y_hat = model(X)
    error = loss_fn( y_hat, y )
    g = torch.autograd.grad(
        outputs = y_hat.sum(),
        inputs  = X,
        grad_outputs = torch.ones(y_hat.size()),
        create_graph = True,
        retain_graph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

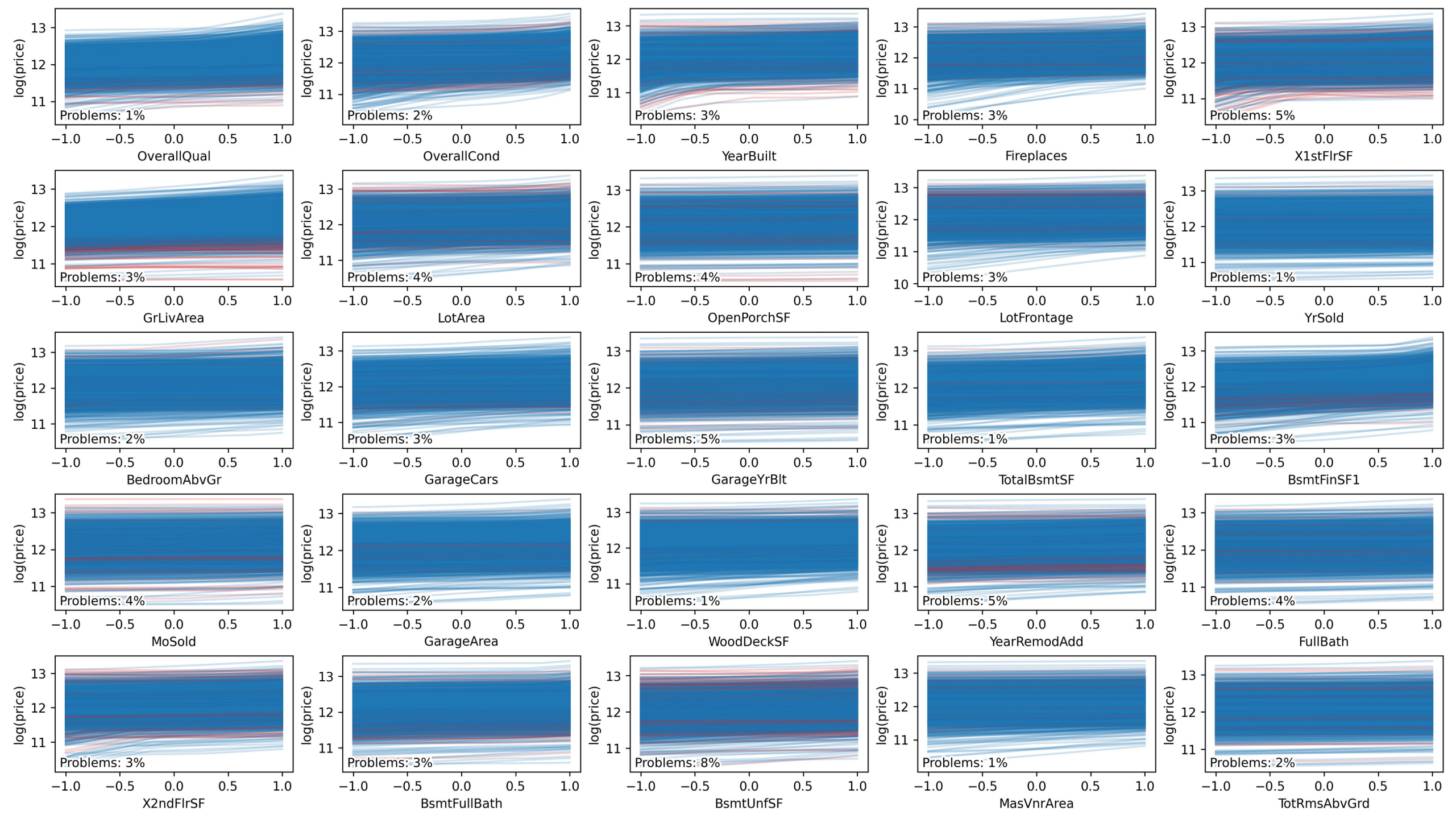
Code: gradient penalty

```
# 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.

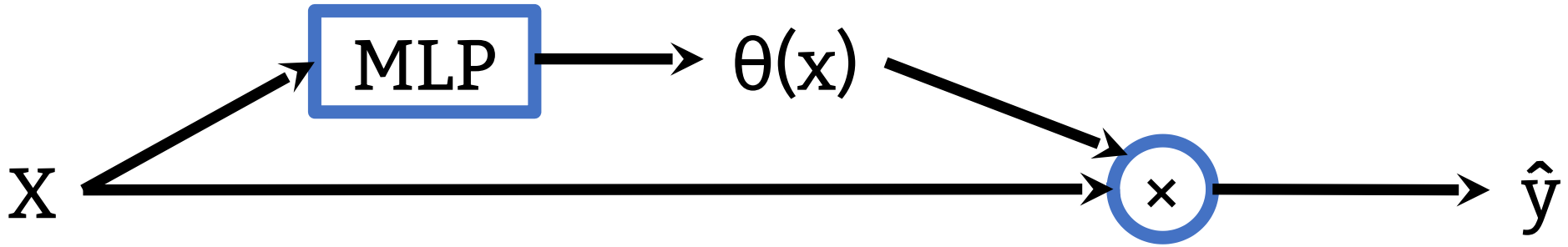
Often, θ only depends on a small number (one) of the x_i's.

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

Varying coefficient models

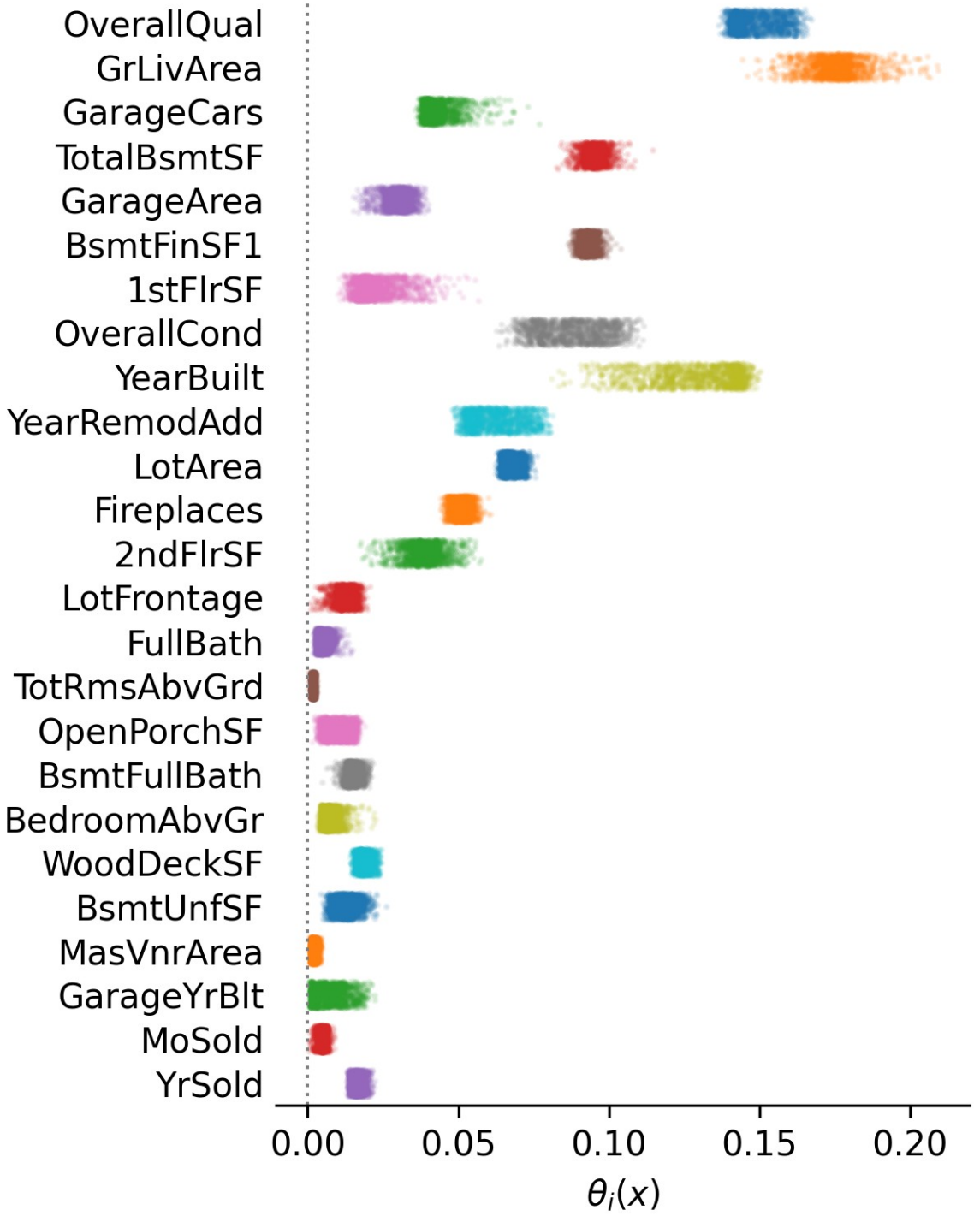
$$\text{Prediction loss: } \|y - \hat{y}\|^2$$

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



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.

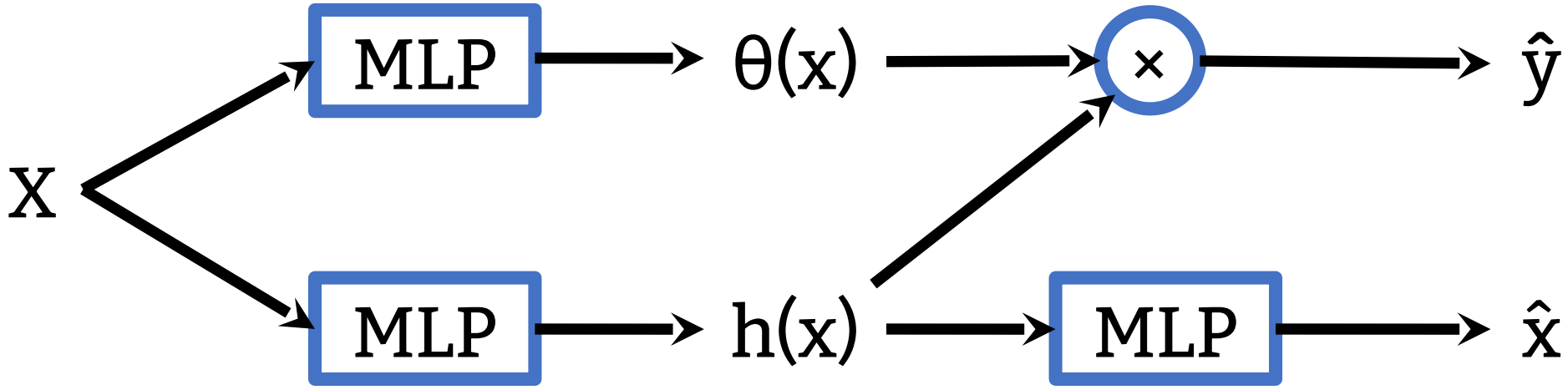
Towards Robust Interpretability with Self-Explaining Neural Networks (D. Alvarez-Melis and T.S. Jaakkola, 2018)

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

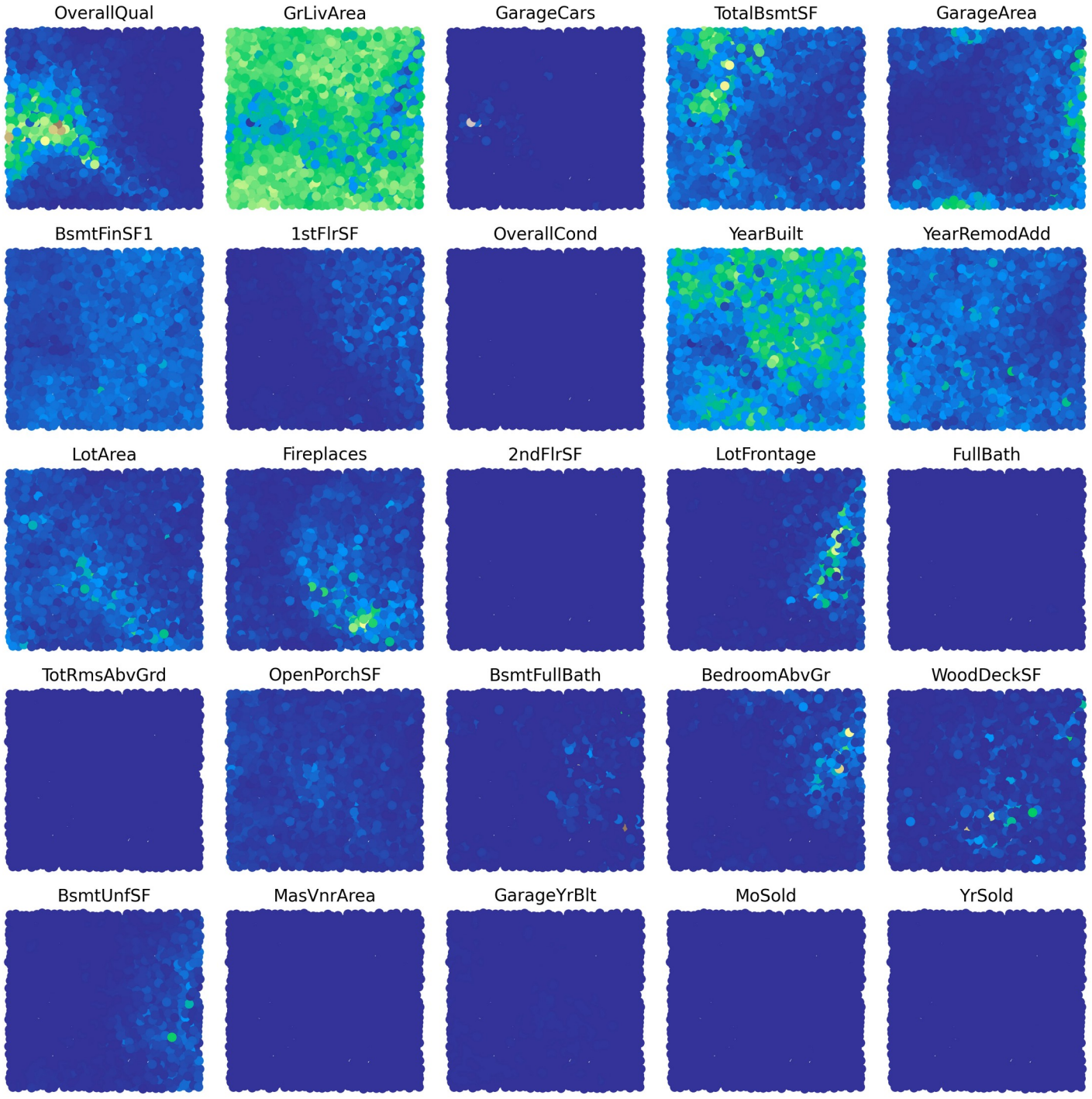
Granger-causal attentive mixtures of experts (P. Schwab et al., 2018)

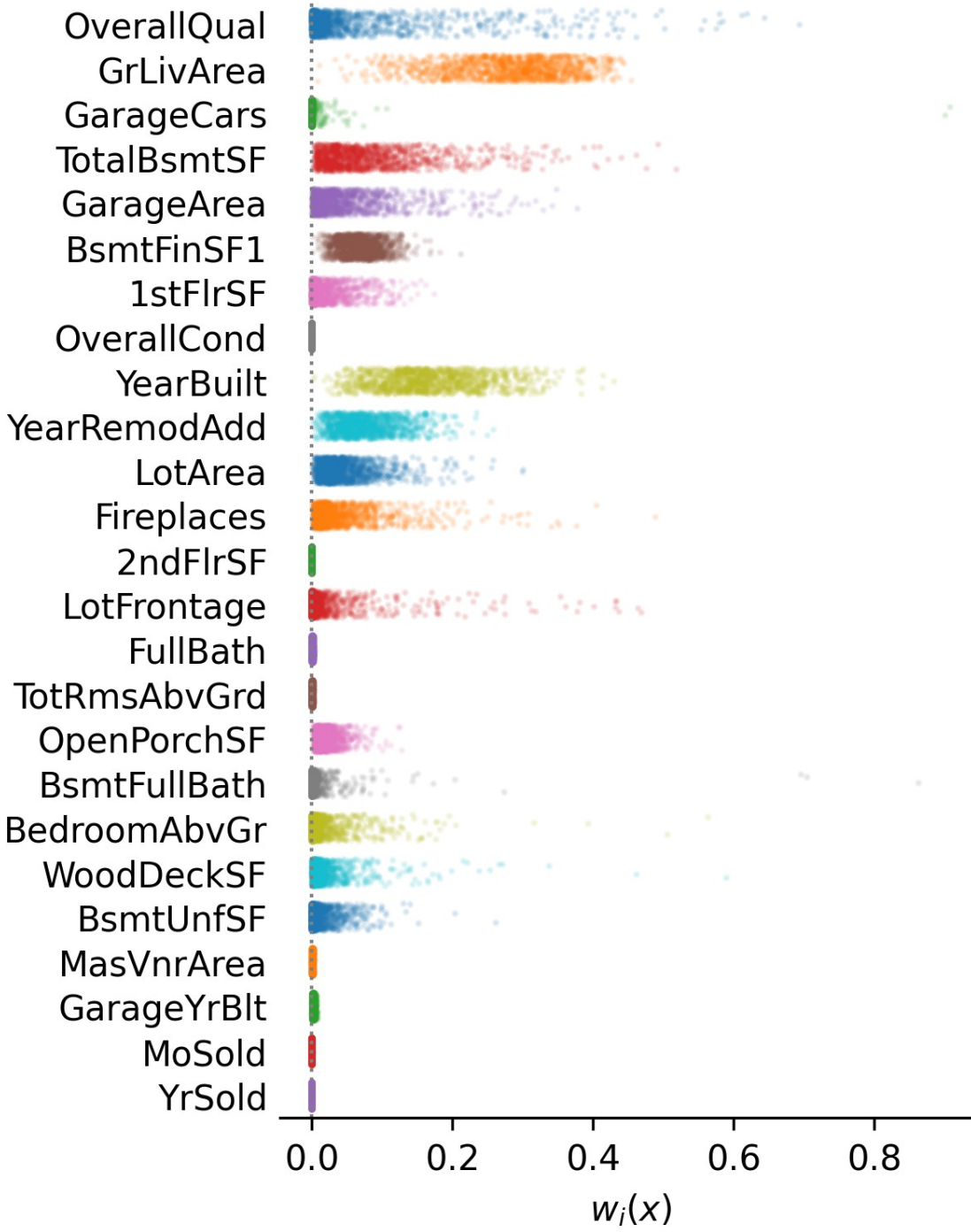
Mixture of experts

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

- Penalty to make $f_i(x_i)$ a predictor of y
- Penalty to make the f_i 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
Lasso	0.023
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

Model	MSE
GAM (monotonic)	**0.017**
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

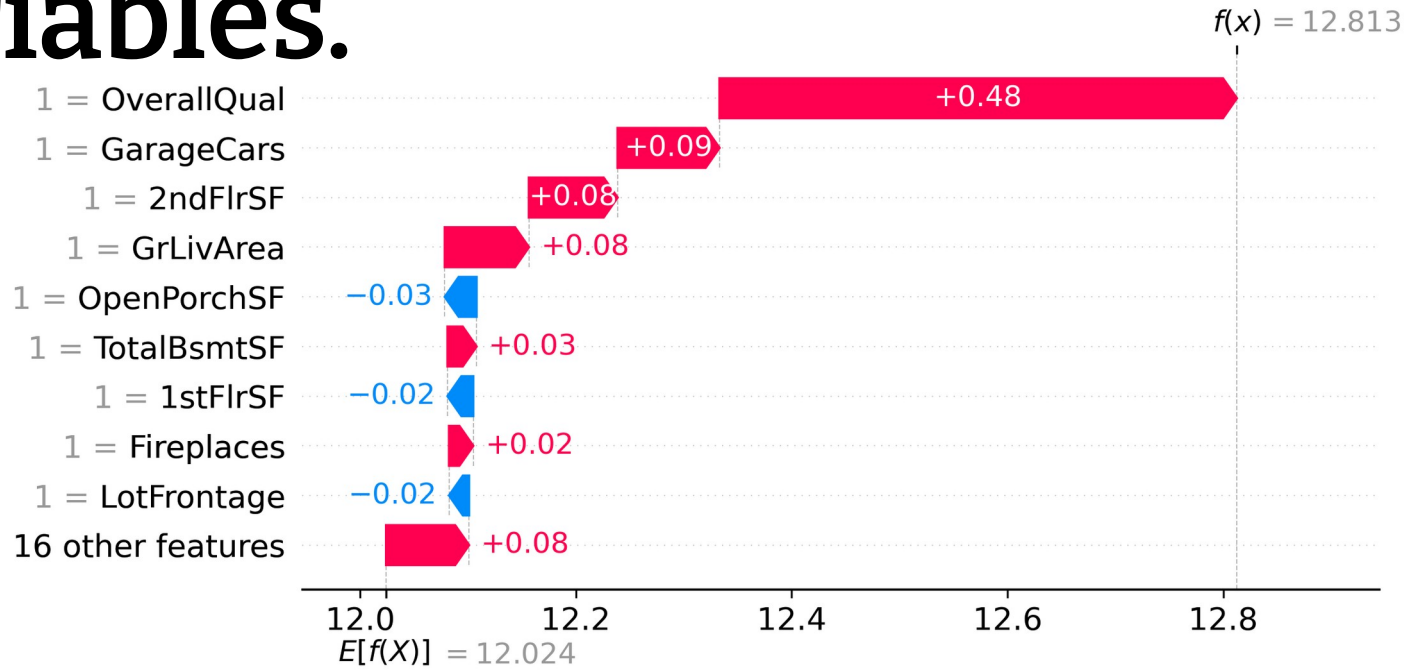
XAI

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)

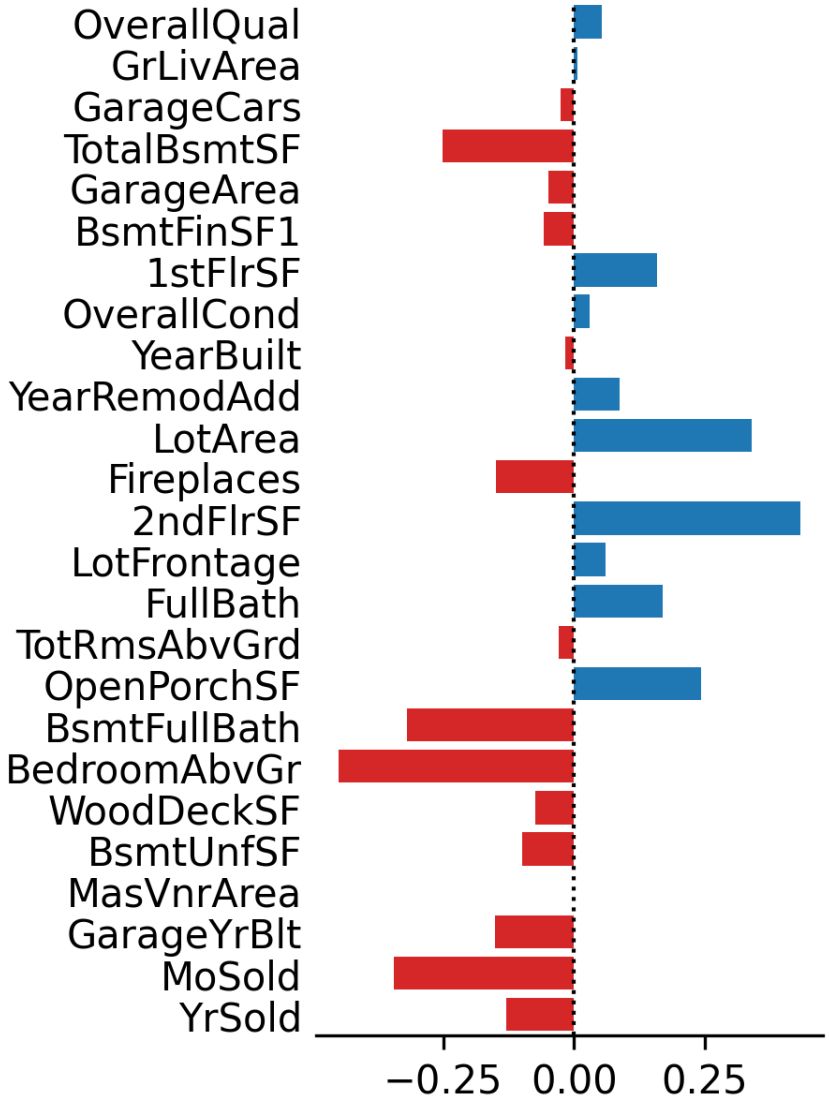
```
# Untested -- do not use

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)}" )
```

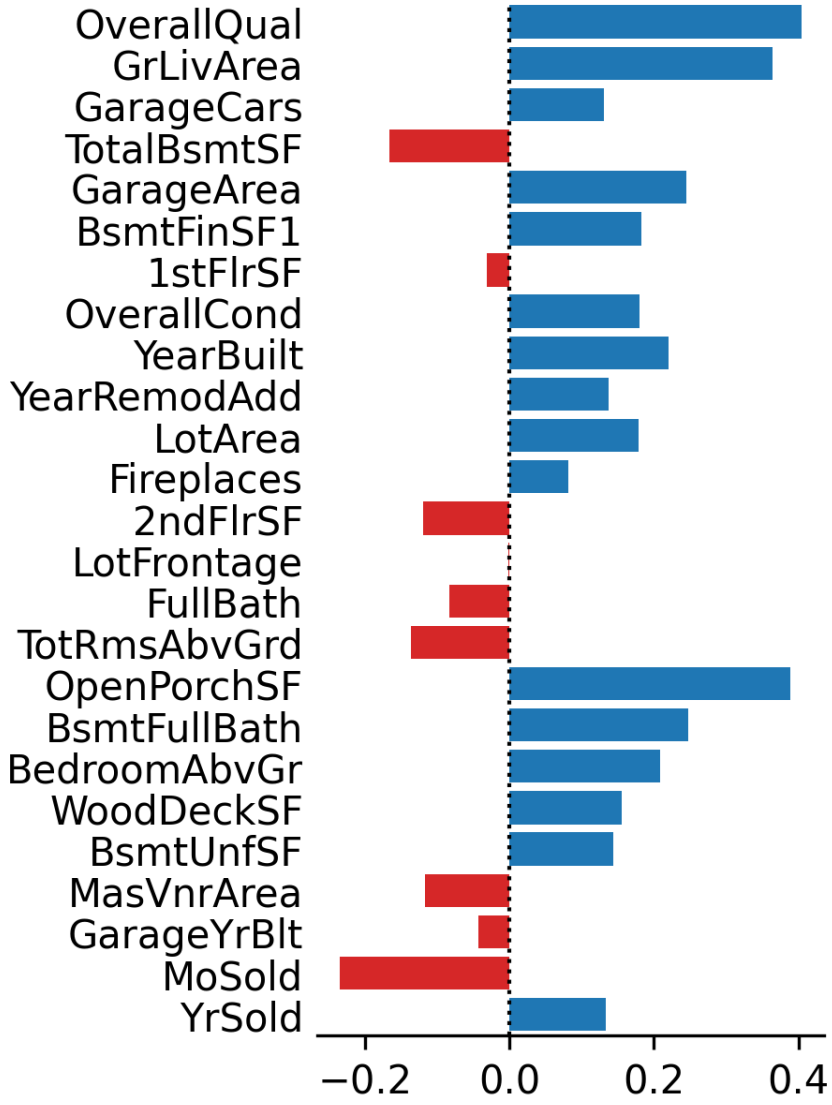
Gradient boosting libraries

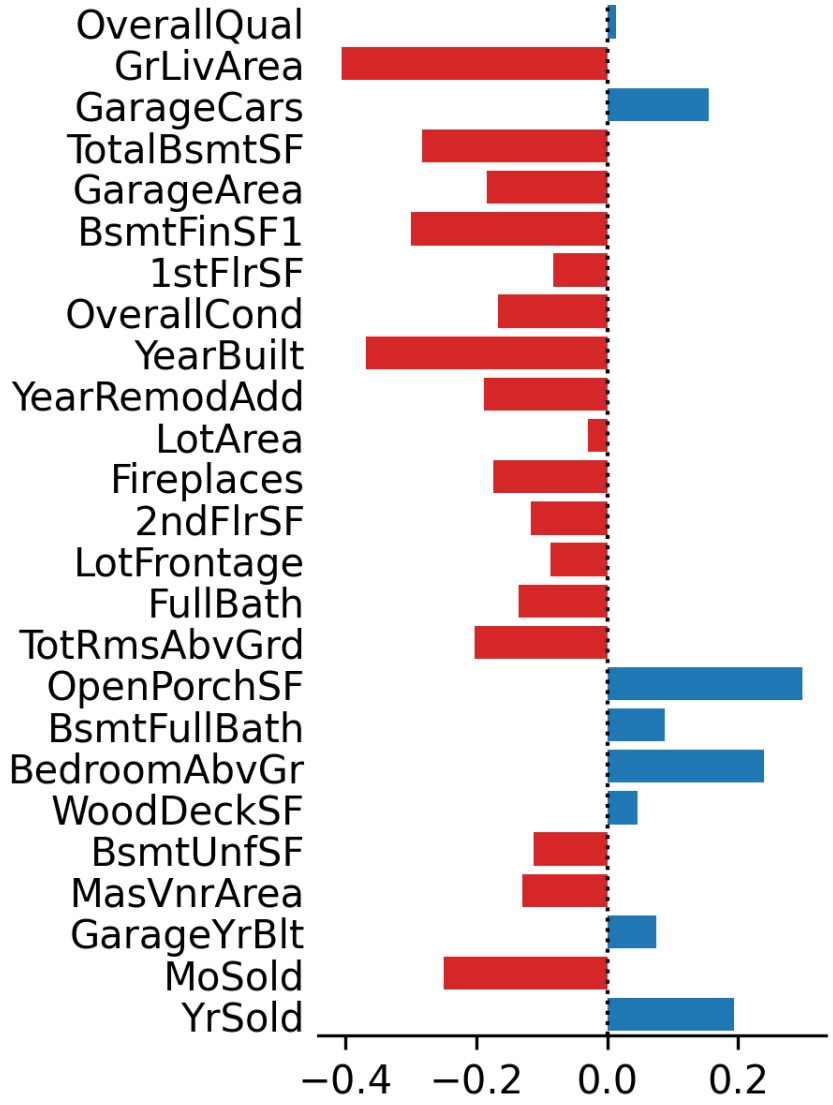
If in doubt, prefer catboost:

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

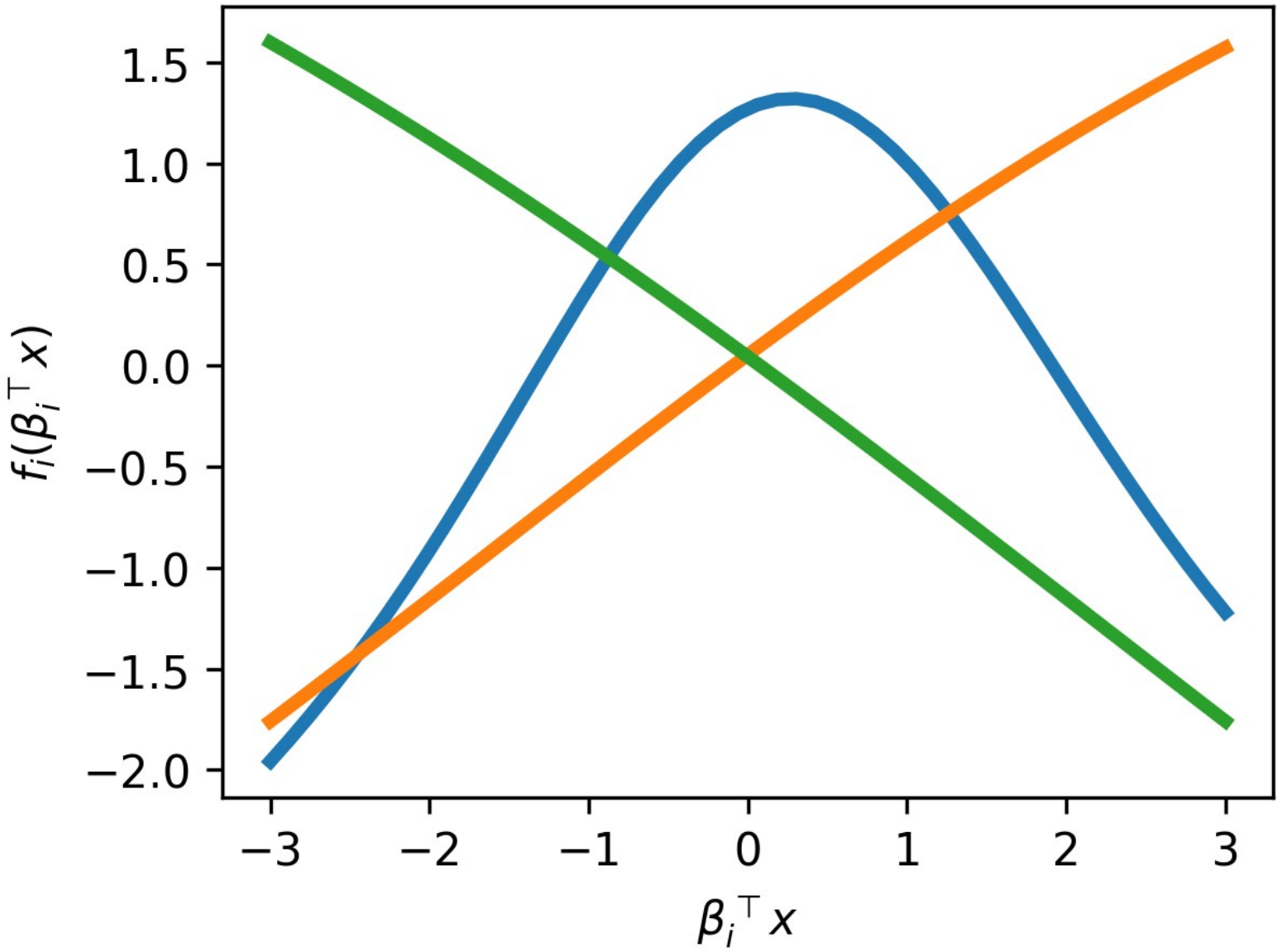
Additive Index Model (AIM)



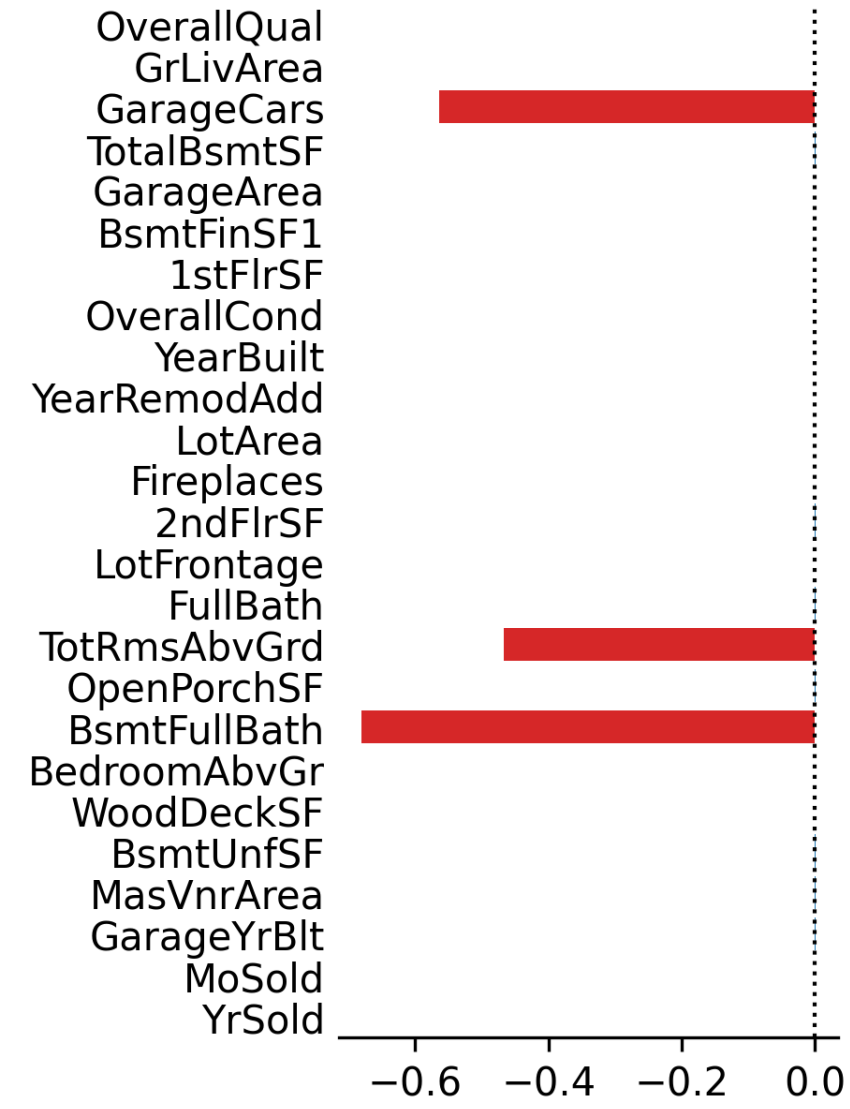


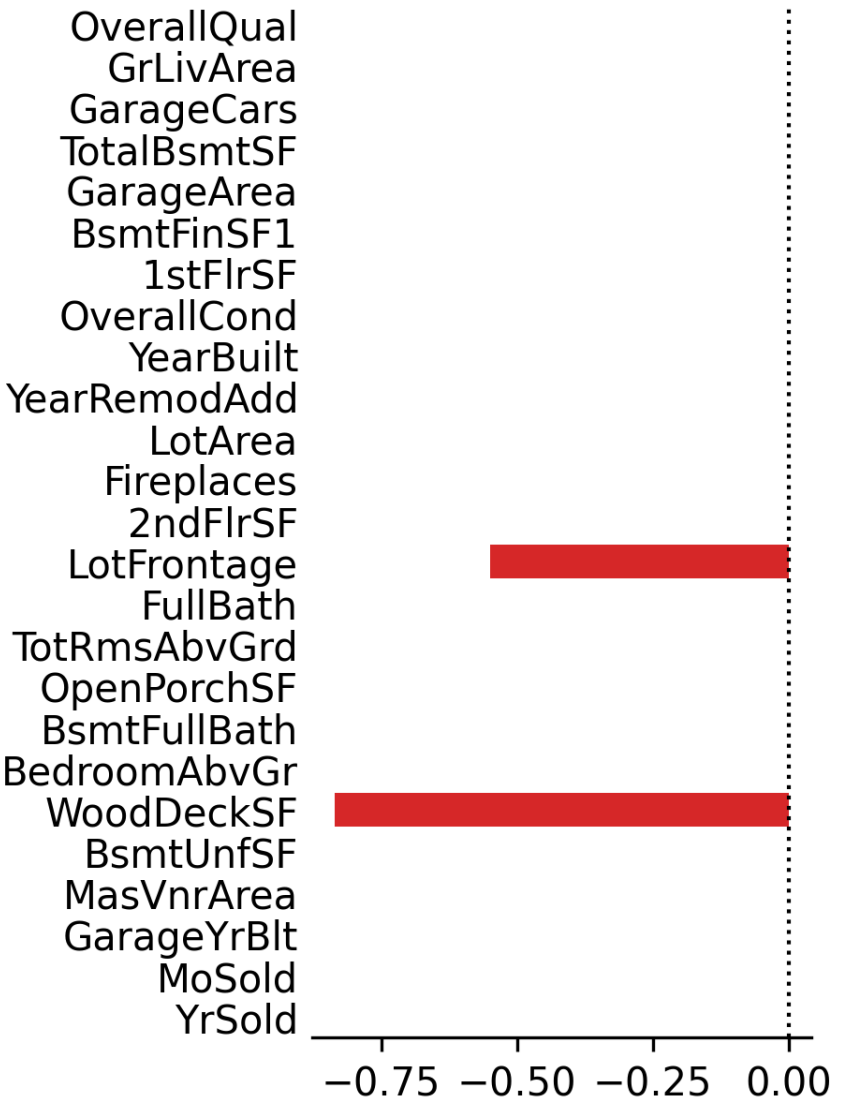


Additive Index Model (AIM)

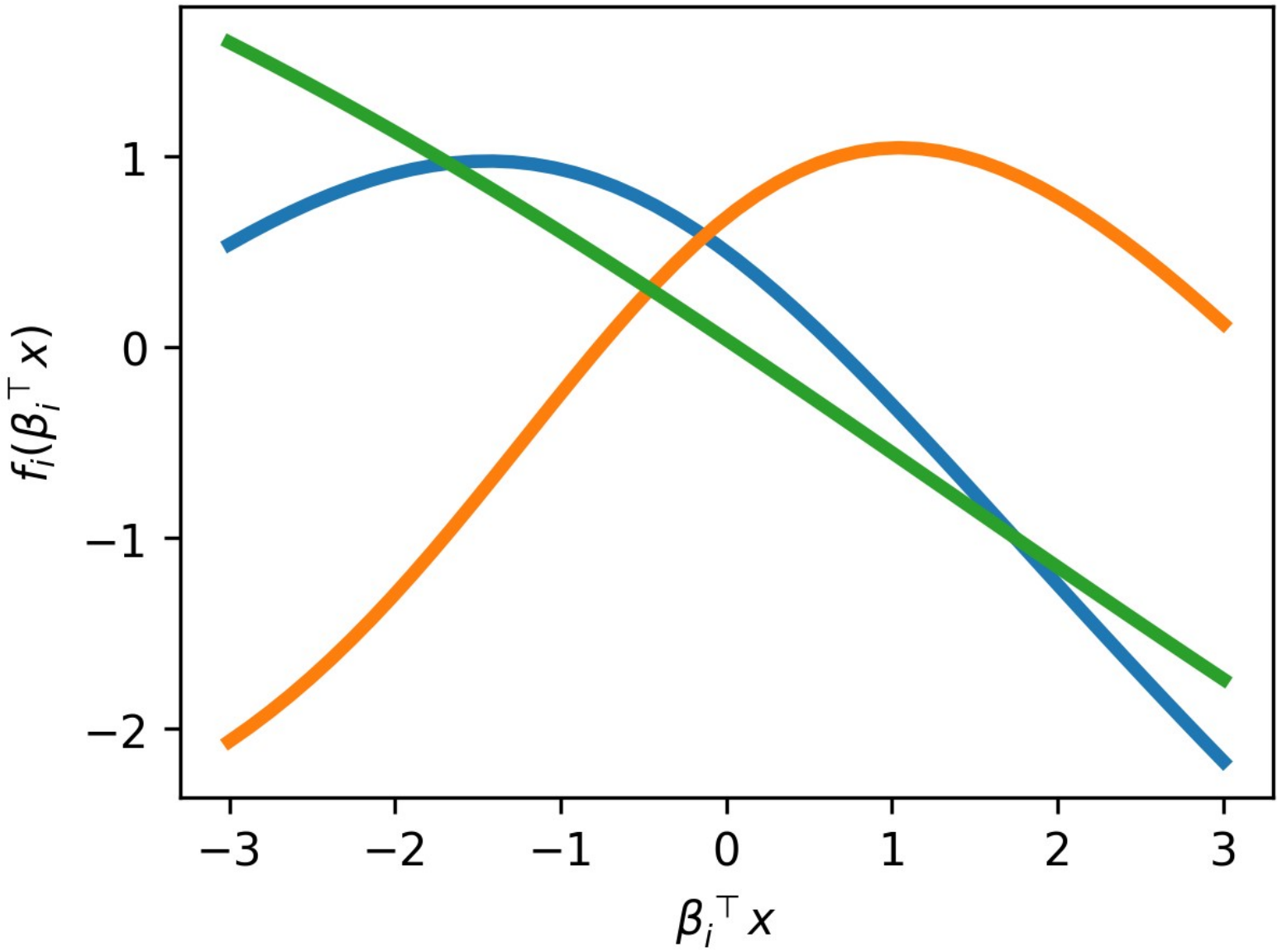


Additive Index Model (AIM)





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:

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

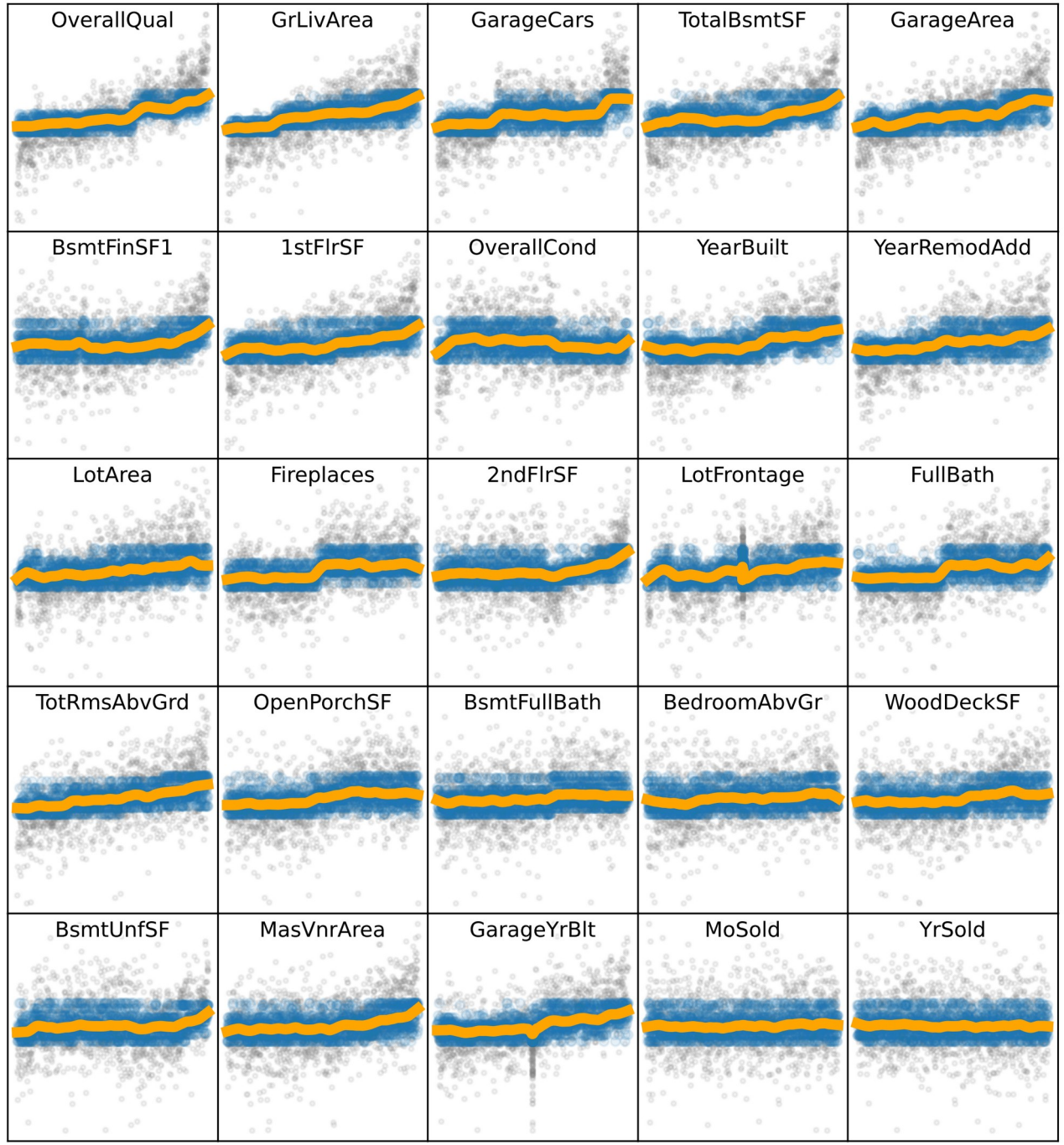
Proximal gradient descent, to minimize $f+g$:

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

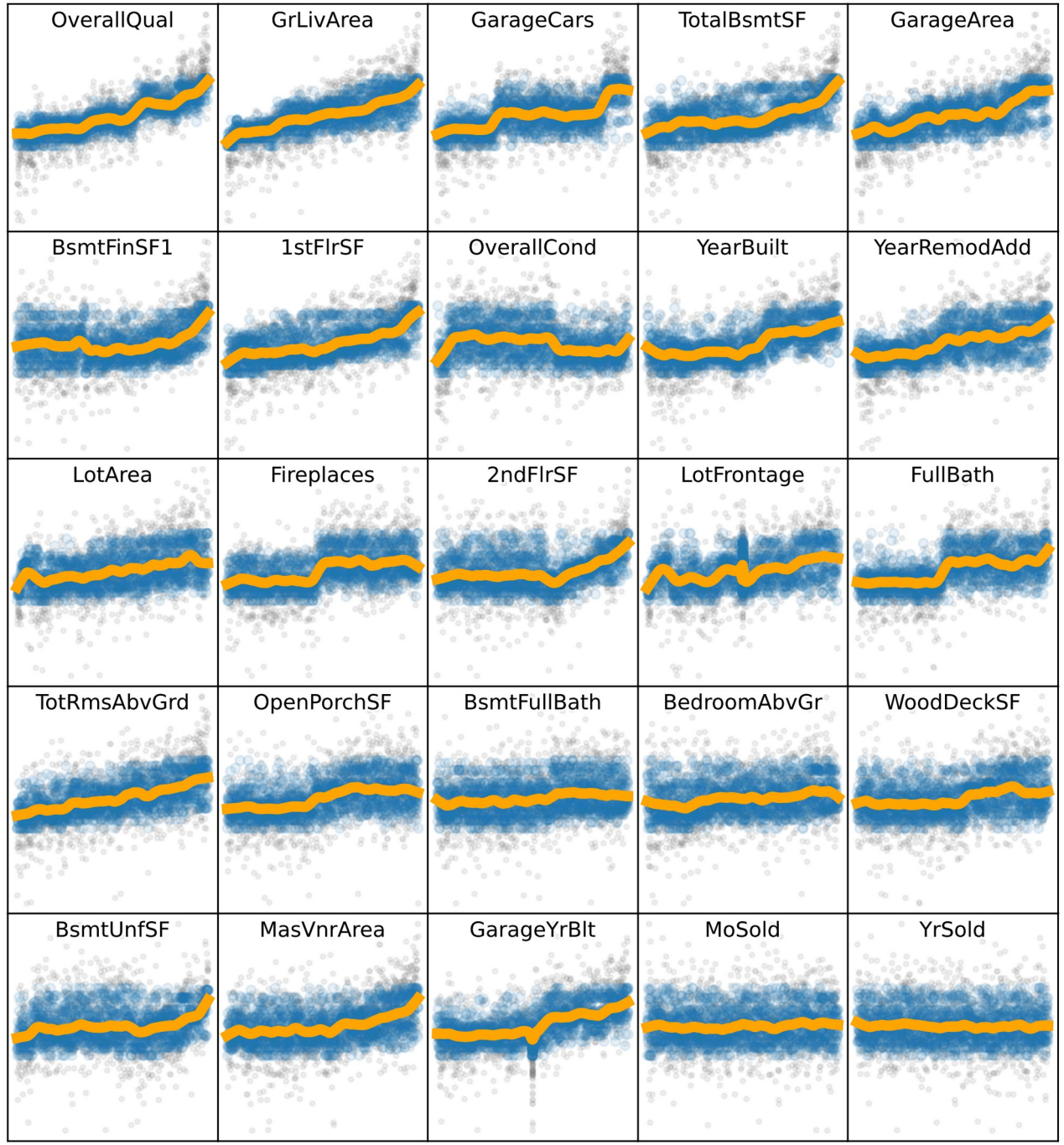
A sparse-group lasso (N. Simon et al., 2013)

Regularization paths

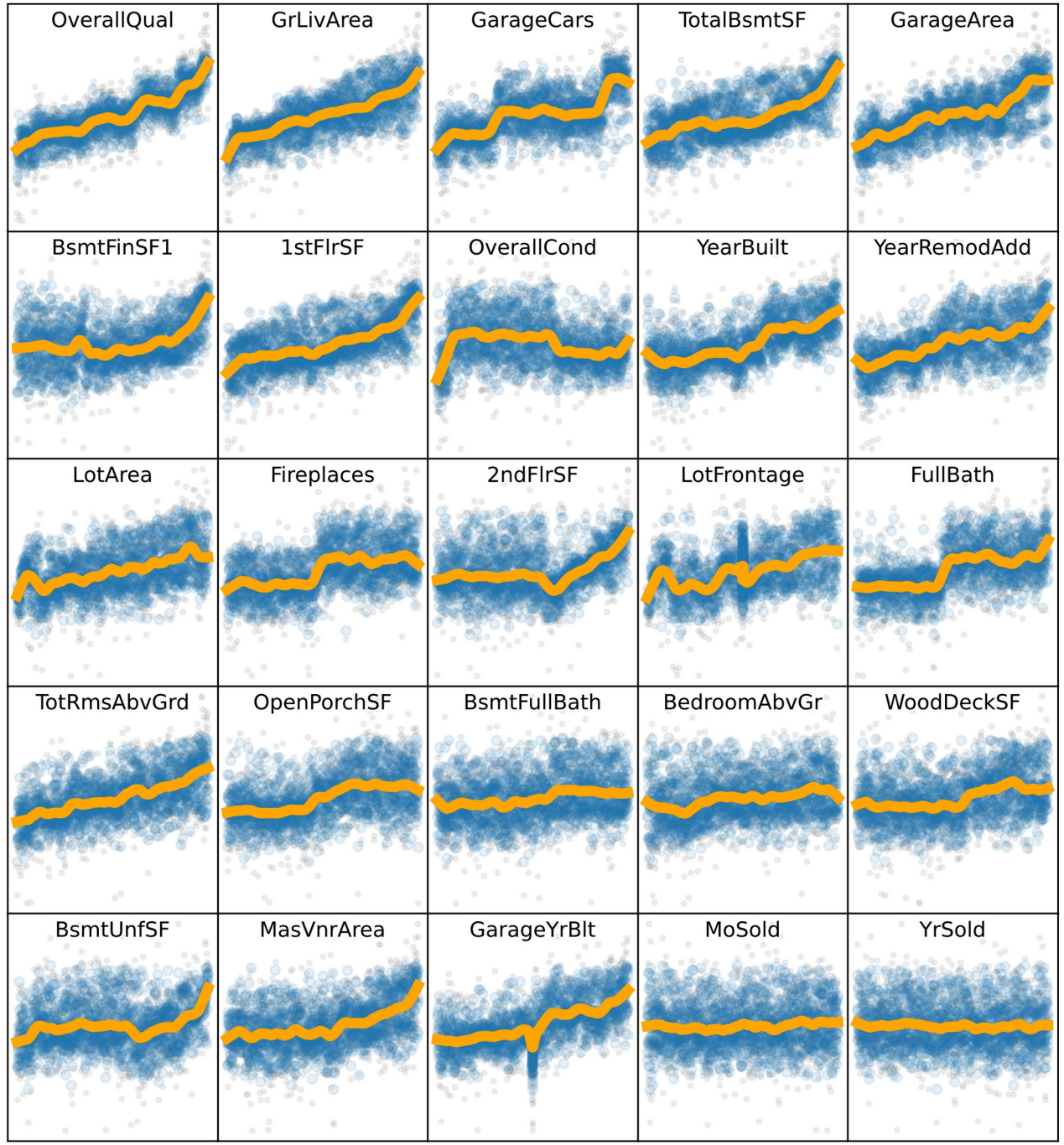
CatBoostRegressor k=1



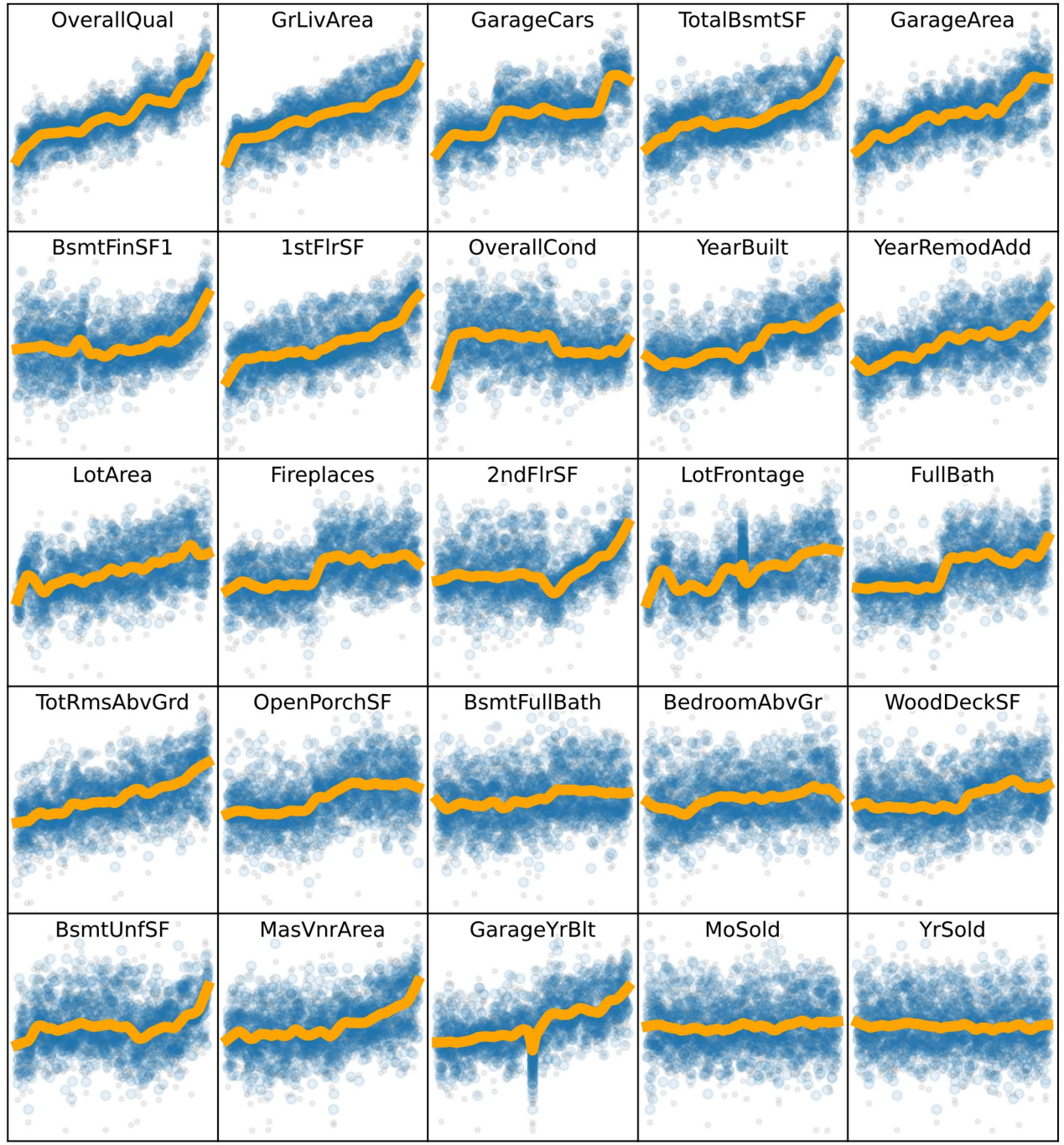
CatBoostRegressor k=2



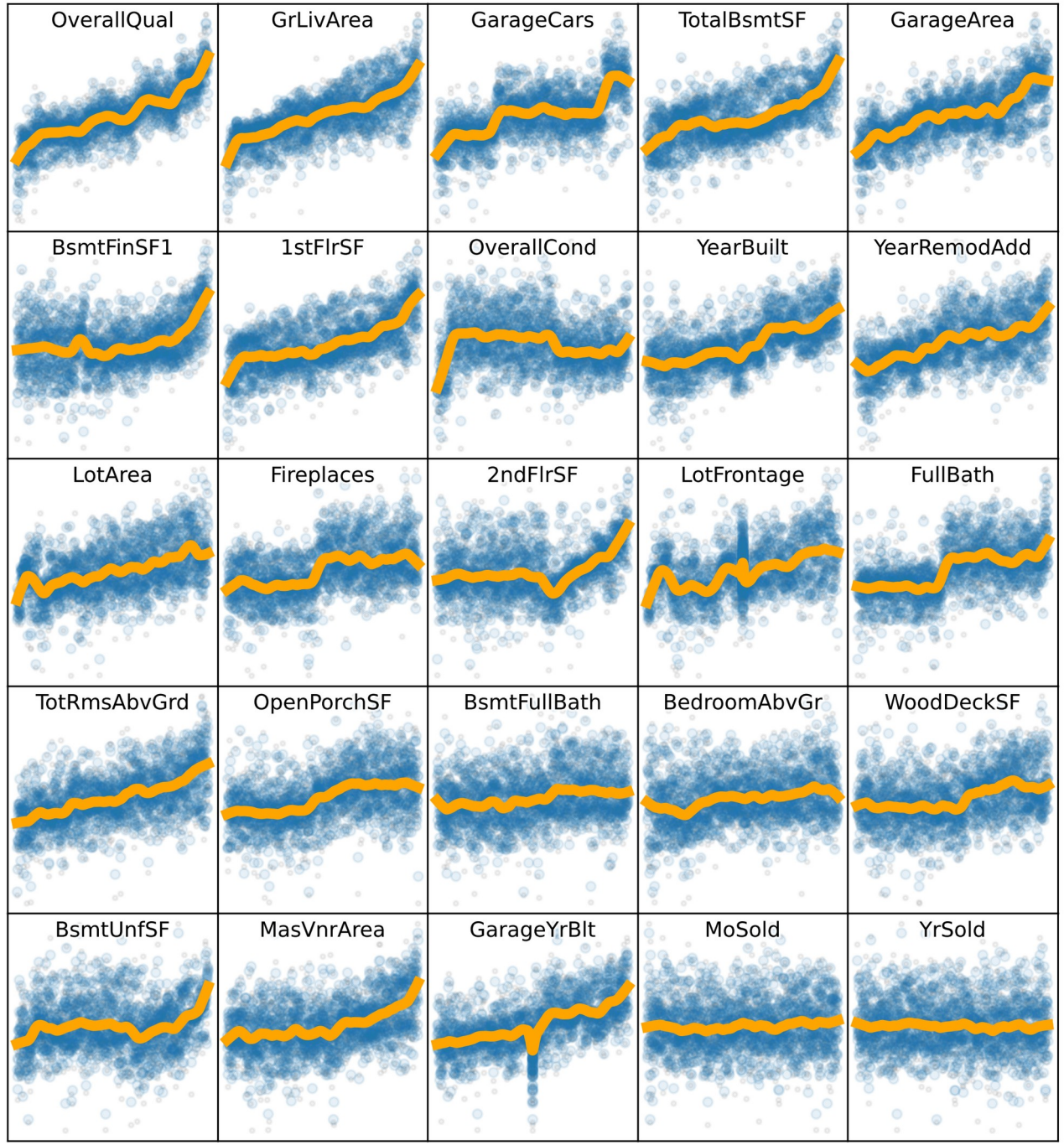
CatBoostRegressor k=5



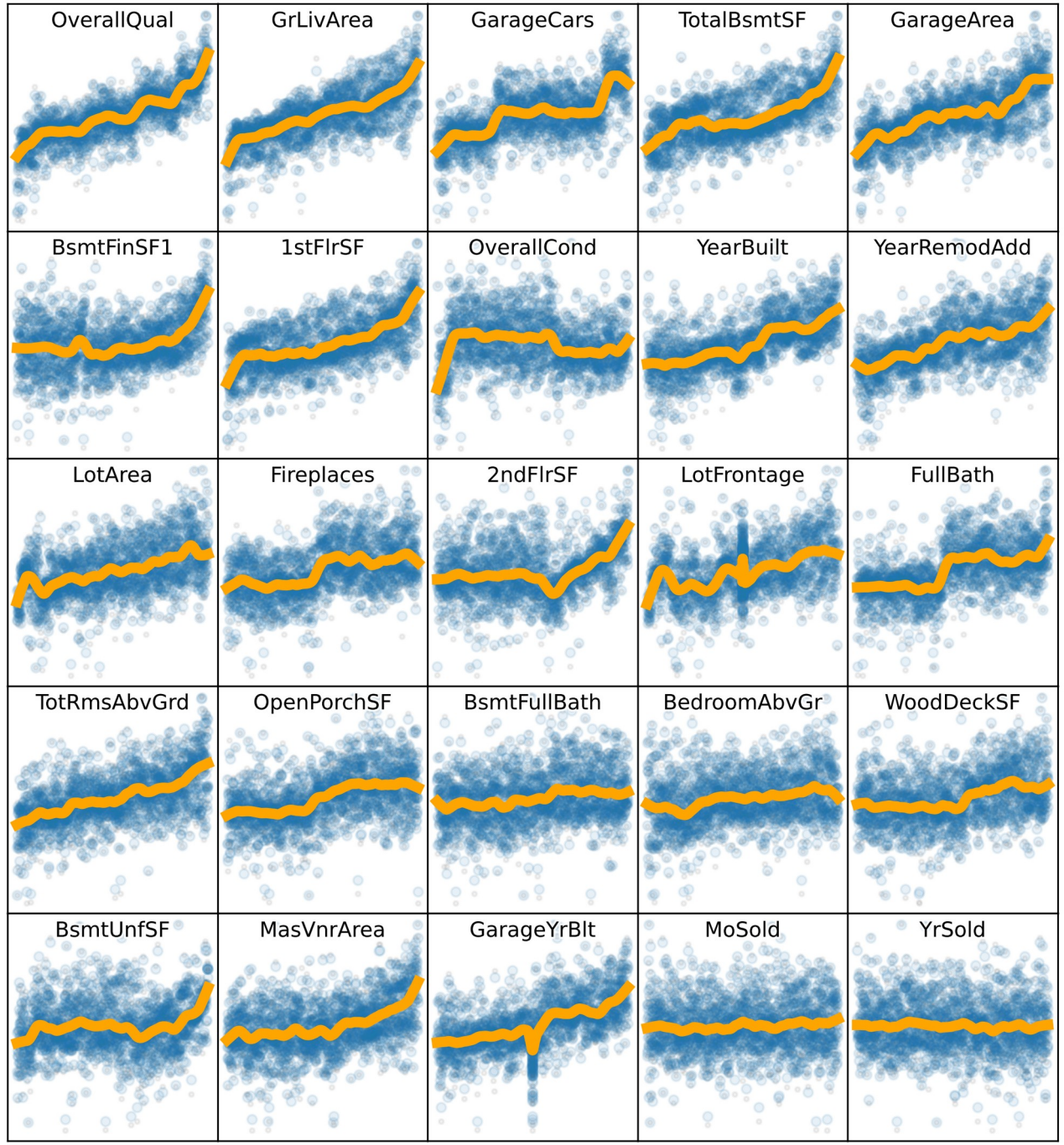
CatBoostRegressor k=10



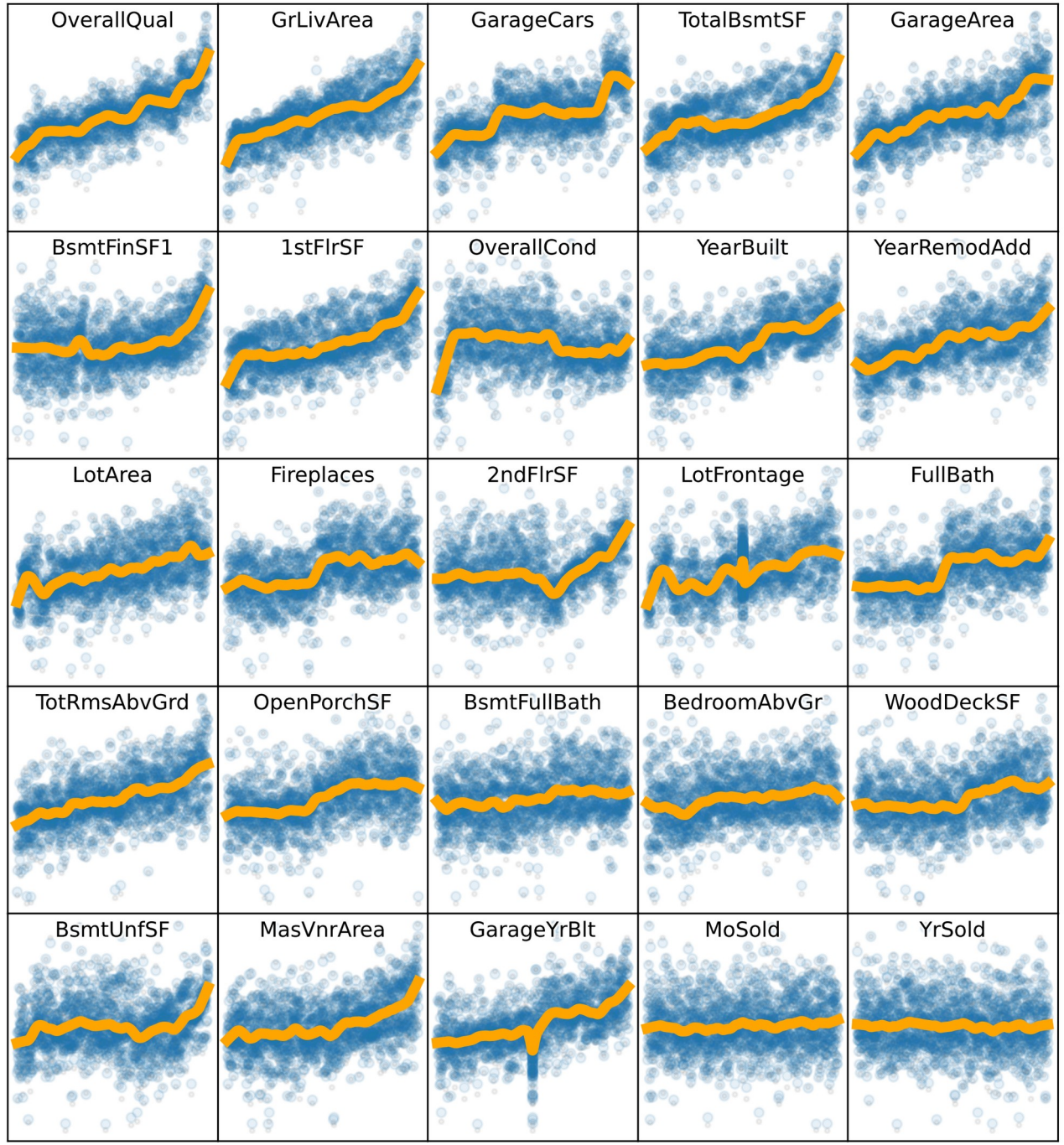
CatBoostRegressor k=20



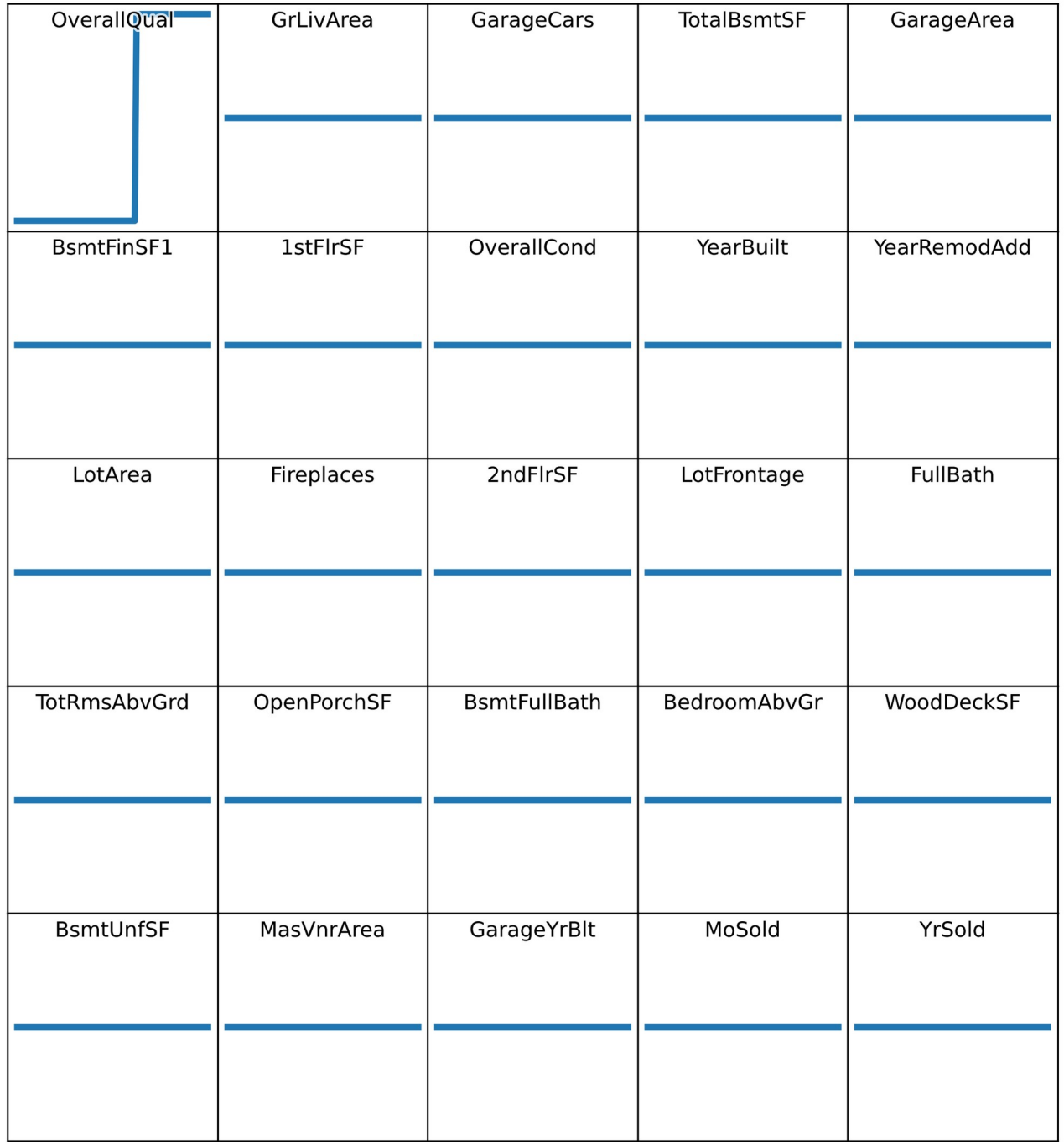
CatBoostRegressor k=50



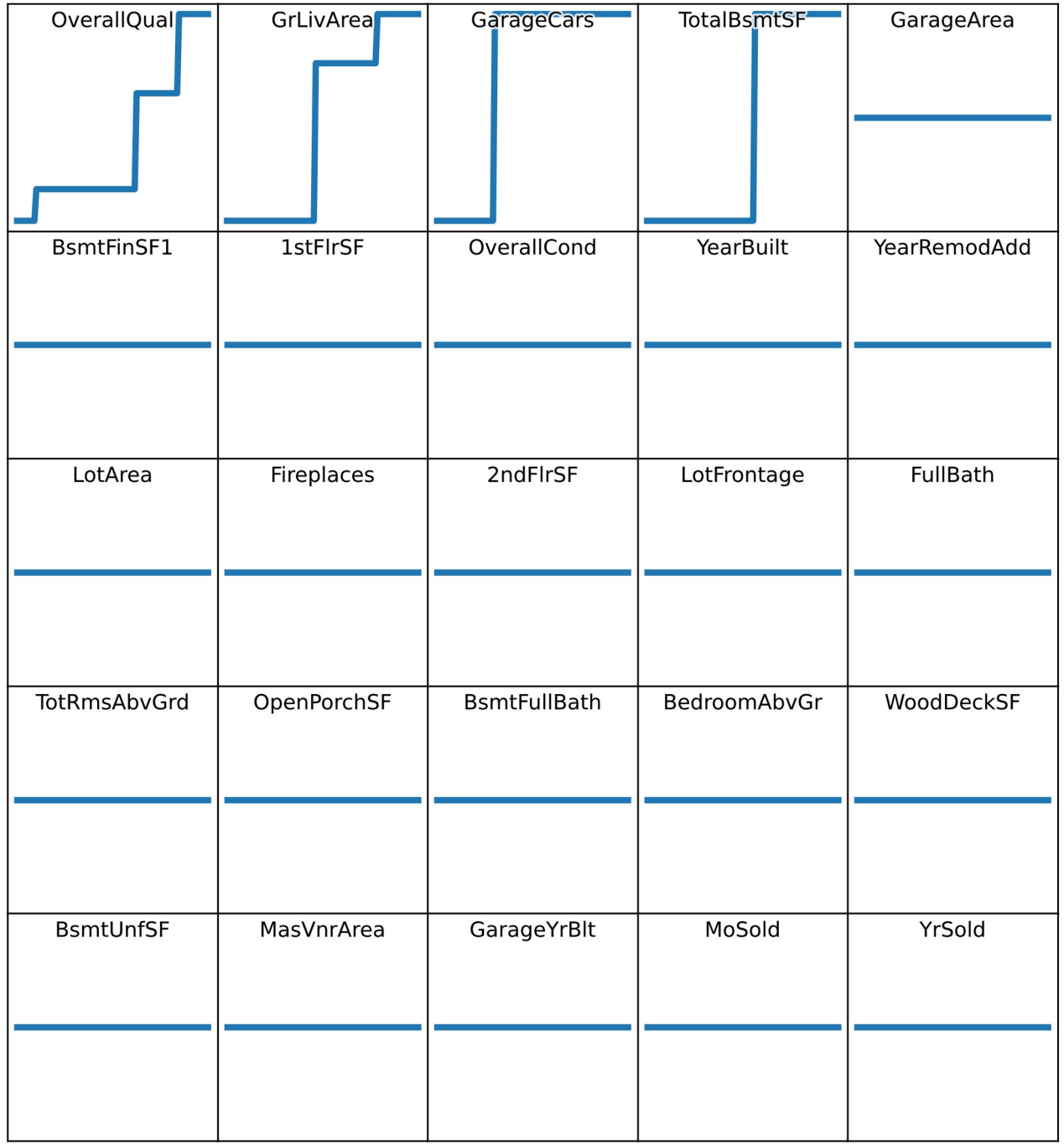
CatBoostRegressor k=100



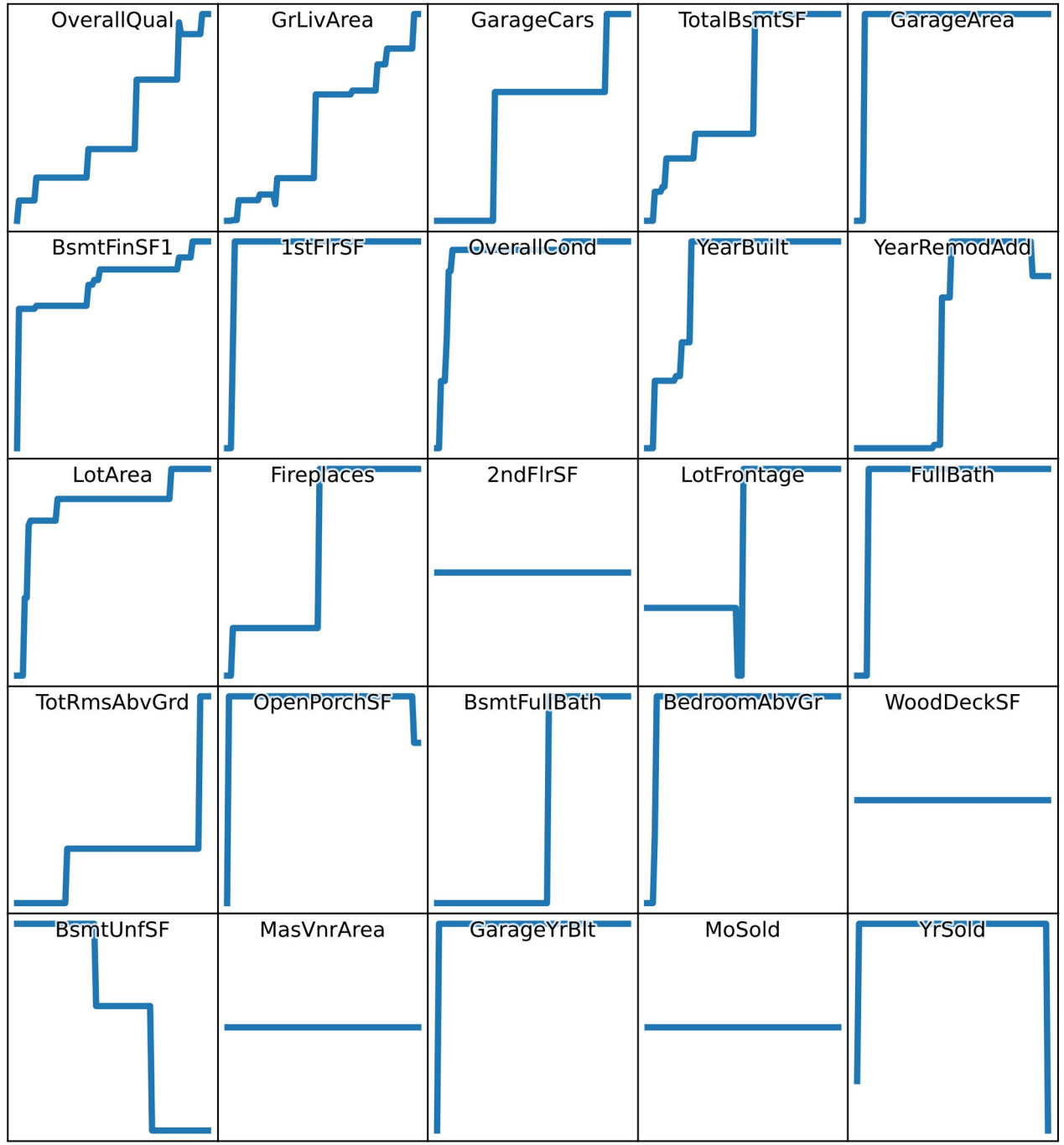
XGBRegressor k=2



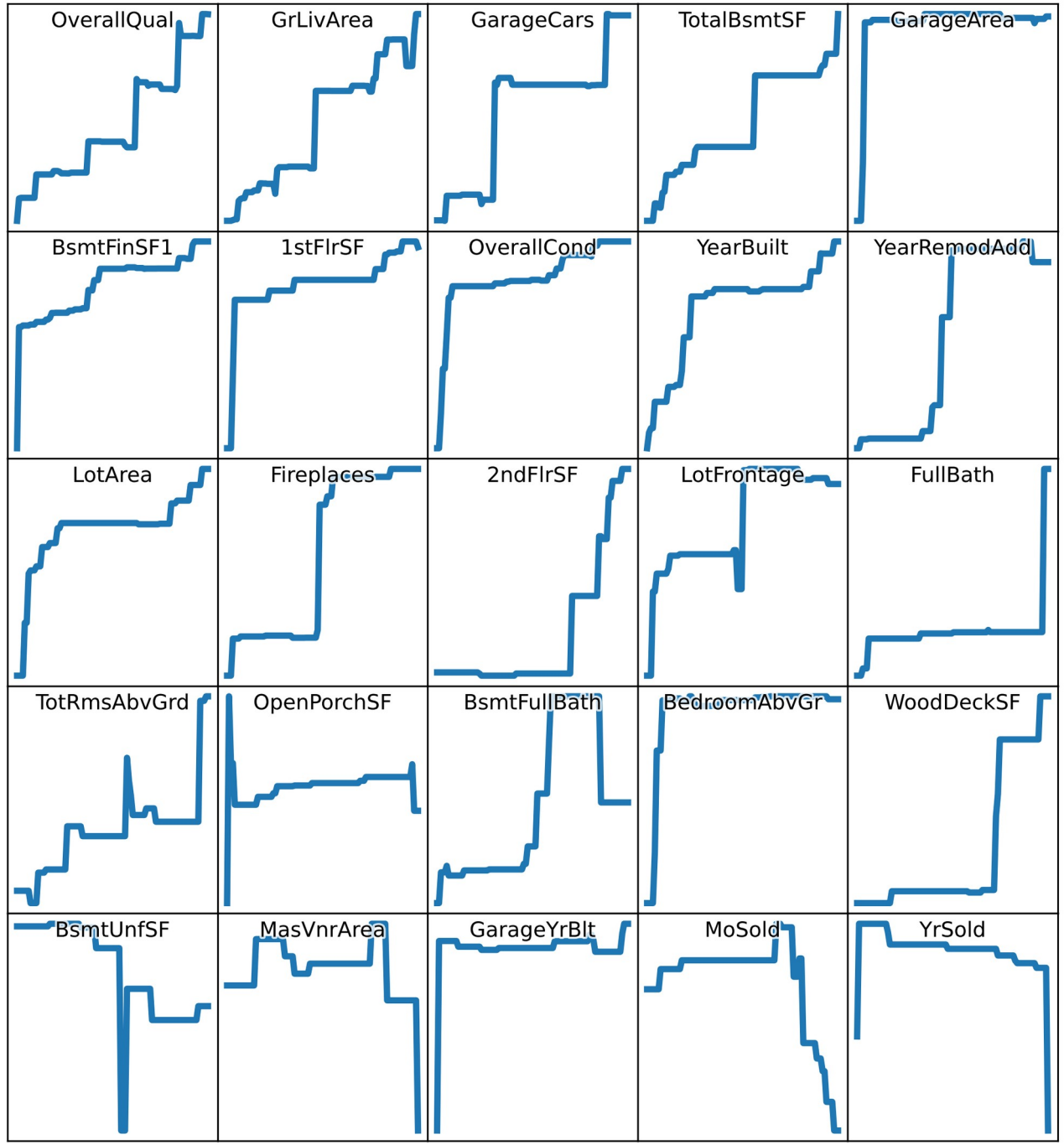
XGBRegressor k=5



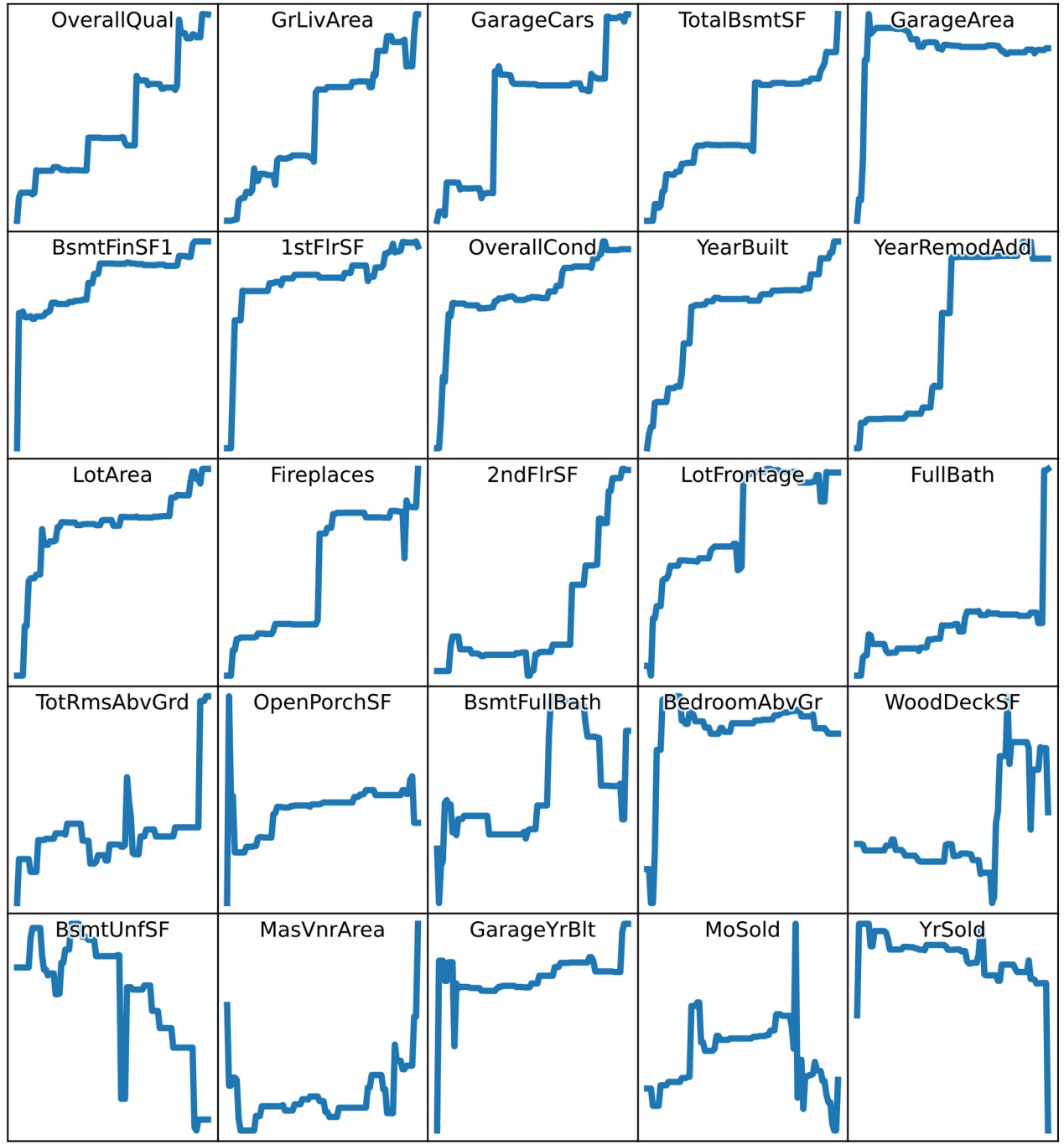
XGBRegressor k=10



XGBRegressor k=20



XGBRegressor k=50



XGBRegressor k=100