NYU

Introduction to Data Science

Center for Data Science
Iddo Drori, Spring 2019

Agenda

- Stochastic Gradient Descent
- Feature extraction and selection
- Random Forest
- Gradient Boosting

Stochastic Gradient Descent

Gradient Descent

Minimize function $f(x_1, \cdots, x_n)$

All first derivatives are 0 at minimum $\frac{\partial f}{\partial x_i} = 0$

Steepest direction in which function decreases fastest is given by gradient

$$x_{t+1} = x_t - \alpha \underbrace{\nabla f(x_t)}_{\text{gradient vector}}$$

Gradient Descent

$$\underset{x \in D}{\text{minimize}}\, f(x)$$

Algorithm 1 Gradient descent

```
given a starting point $x \in \textbf{dom} f$
repeat
    determine descent direction $\Delta x = -\nabla f(x)$
    choose step size $\alpha$
    update $x = x + \alpha \Delta x$
until stopping criterion is satisfied.
```

Gradient Descent

- Fundamental in training neural networks
- Minimize total loss function $\mathcal{L}(x)$
- Based on step $x_{t+1} = x_t - \alpha \nabla \mathcal{L}(x_t)$

gradient vector

- Problems:
  At every descent step computing $\nabla \mathcal{L}$

1. Total loss adds individual losses of every training sample: many examples

$$\mathcal{L}(x) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(x, a^{(i)})$$

weights x examples a

2. Total loss with respect to all the weights: many weights

Stochastic Gradient Descent

- Solutions: at each step

1. Use only minibatch of training data, random sample(s)

2. Backpropagation: compute derivative with respect to all weights in a single backward pass.

Loss Functions

- Sum of errors in classifying each of the training examples
- Describe learning function which classifies each example

- Loss functions:

1. Square loss
$$\mathcal{L}(x) = \frac{1}{m}\sum_{i=1}^{m}\left\|f\left(x, a^{(i)}\right) - y^{(i)}\right\|^2$$
weights x
examples a
ground truth labels y

2. Hinge loss
$$\mathcal{L}(x) = \frac{1}{m}\sum_{i=1}^{m}\max\left(0, 1 - tf(x)\right)$$
$t=1$ or $t=-1$ for classification

3. Cross-entropy loss
$$\mathcal{L}(x) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log\widehat{y^{(i)}} + \left(1 - y^{(i)}\right)\log\left(1 - \widehat{y^{(i)}}\right)$$
prediction $\hat{y}$

- For mini-batch of size k, $m=k$ random samples

NYU

Stochastic Gradient Descent

- Mini-batch with 1 random sample

$$x_{t+1} = x_t - \alpha \underbrace{\nabla_x \mathcal{L}}_{\text{derivative of loss term}}\big(x_t, \underbrace{a^{(i)}}_{\text{sample}}\big)$$

- Random ordering of training data
- Epoch: pass through training data
- Early steps often converge faster to solution than gradient descent

NYU

Mini-Batch Gradient Descent

Split data into n/k disjoint sets of size k

Instead of process entire dataset, iteratively process each of the subsets

Example:
estimating gradient from 100 samples instead of 10000 samples

+ Reducing computation time and memory by a factor of **100**
- Reduces standard error of the mean only by a factor of **10**:
    - Standard error of sample mean is $s/sqrt(n)$ where s is the standard deviation of the population and n in the number of observations of the samples.
    - If $x1,\dots,x_n$ are n independent sample then the variance of their sum $T=sum(xi)$ is $ns\^2$
    - The variance of the sample mean T/n is $s\^2/n$ and the standard deviation is $s/sqrt(n)$

NYU

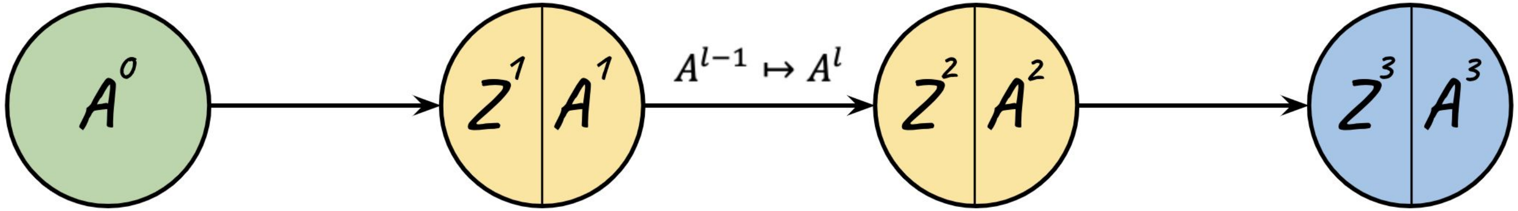
Adaptive Stochastic Gradient Descent

- Faster convergence
- Use gradients from earlier steps

Forward Propagation

$$f(x) = f^3\left(f^2(f^1(x))\right)$$

composition of functions



$$A^l = f^l(A^{l-1})$$

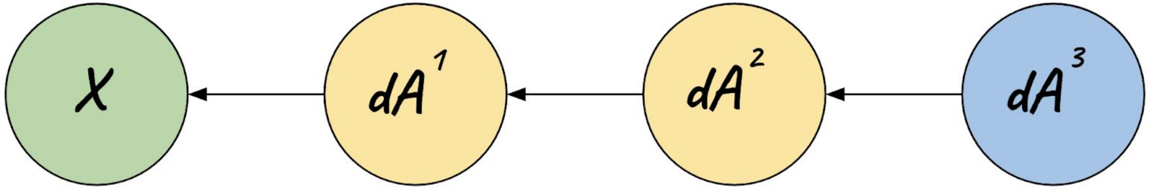
$$Z^l = W^l A^{l-1} \qquad A^l = g^l(Z^l)$$

linear non-linear

$$A^3 = g^3\left(W^3 g^2(W^2 g^1(W^1 A^0))\right)$$

if g is identity then f is linear in x

Backpropagation



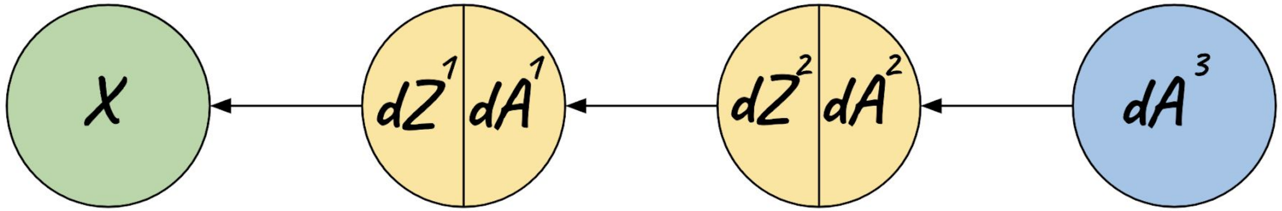
Map derivatives from layer *l* back to layer *l-1* with respect to **activations** and **weights**

$$dA^l \mapsto dA^{l-1}$$

$$dA^l := \frac{d\mathcal{L}}{dA^l} \qquad dZ^l := \frac{d\mathcal{L}}{dZ^l} \qquad dW^l := \frac{d\mathcal{L}}{dW^l}$$

NYU

Backpropagation



Map derivatives from layer *l* back to layer *l-1* with respect to **activations**

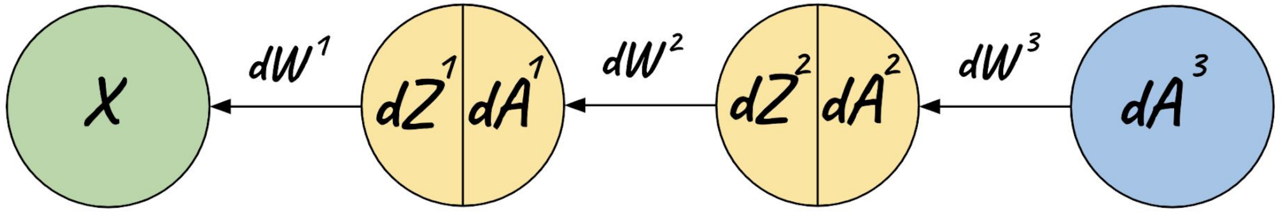
$$dA^{l-1} = \left(W^{l}\right)^{T} dZ^{l}$$

From chain rule of differentiation

$$\frac{d\mathcal{L}}{dZ^{l}} = \frac{d\mathcal{L}}{dA^{l}}\frac{dA^{l}}{dZ^{l}} = \frac{d\mathcal{L}}{dA^{l}} \cdot g^{l\prime}(Z^{l})$$

$$dZ^{l} = dA^{l} \cdot g^{l\prime}(Z^{l})$$

Backpropagation



Map derivatives from layer *l* back to layer *l-1* with respect to **weights**

$$dW^l = dZ^l (A^{l-1})^T$$

Feature Selection

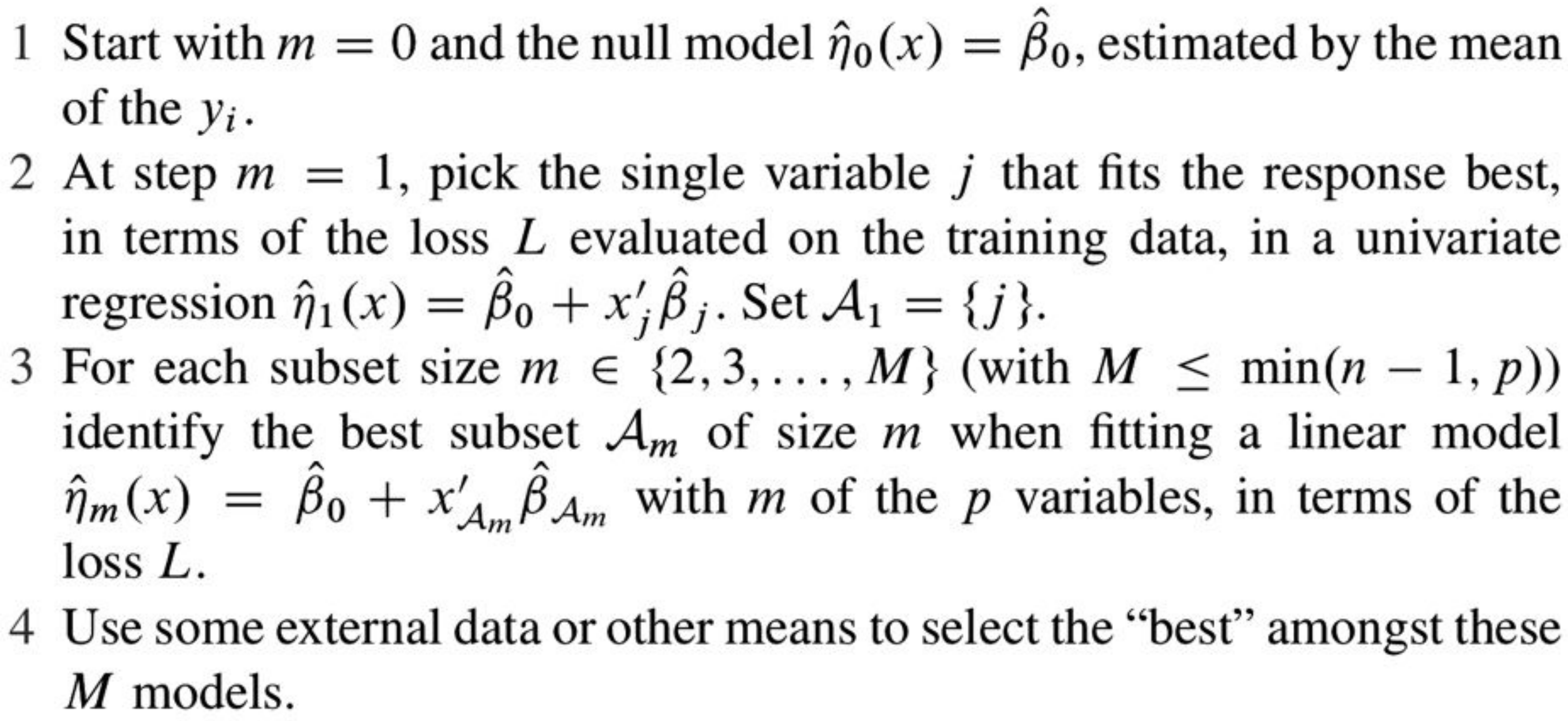
Feature Selection

- Wide datasets: # of features p > # of examples n
  - Genomics: measure expression of many genes
  - Natural language models: word representation

- Identify good subset of features

NYU

Best Subset Regression



1 Start with $m = 0$ and the null model $\hat{\eta}_0(x) = \hat{\beta}_0$, estimated by the mean of the y_i.

2 At step $m = 1$, pick the single variable j that fits the response best, in terms of the loss L evaluated on the training data, in a univariate regression $\hat{\eta}_1(x) = \hat{\beta}_0 + x_j' \hat{\beta}_j$. Set $\mathcal{A}_1 = \{j\}$.

3 For each subset size $m \in \{2, 3, \ldots, M\}$ (with $M \leq \min(n-1, p)$) identify the best subset $\mathcal{A}_m$ of size m when fitting a linear model $\hat{\eta}_m(x) = \hat{\beta}_0 + x_{\mathcal{A}_m}' \hat{\beta}_{\mathcal{A}_m}$ with m of the p variables, in terms of the loss L.

4 Use some external data or other means to select the "best" amongst these M models.

Source: CASI

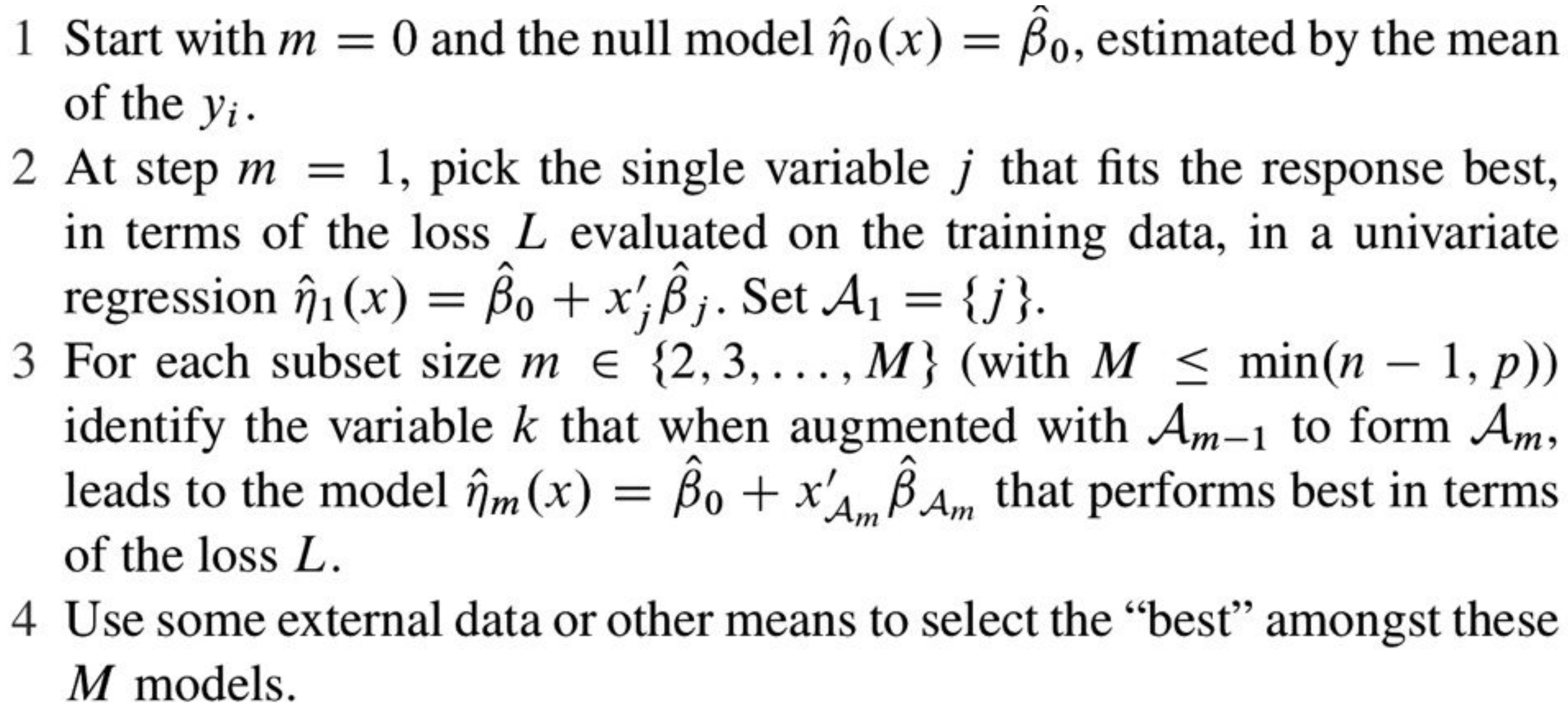
Best Subset

- All subsets: for p features, 2^p combinations
- NP complete

Best Subset Regression

$$\min \|x\|_0 \text{ subject to } y = Ax$$

Forward Stepwise Regression



1 Start with $m = 0$ and the null model $\hat{\eta}_0(x) = \hat{\beta}_0$, estimated by the mean of the y_i.

2 At step $m = 1$, pick the single variable j that fits the response best, in terms of the loss L evaluated on the training data, in a univariate regression $\hat{\eta}_1(x) = \hat{\beta}_0 + x_j'\hat{\beta}_j$. Set $\mathcal{A}_1 = \{j\}$.

3 For each subset size $m \in \{2, 3, \ldots, M\}$ (with $M \leq \min(n-1, p)$) identify the variable k that when augmented with $\mathcal{A}_{m-1}$ to form $\mathcal{A}_m$, leads to the model $\hat{\eta}_m(x) = \hat{\beta}_0 + x_{\mathcal{A}_m}'\hat{\beta}_{\mathcal{A}_m}$ that performs best in terms of the loss L.

4 Use some external data or other means to select the "best" amongst these M models.

Source: CASI

NYU

Forward Stepwise Regression

- Greedy
- Nested subsets
- May not be optimal

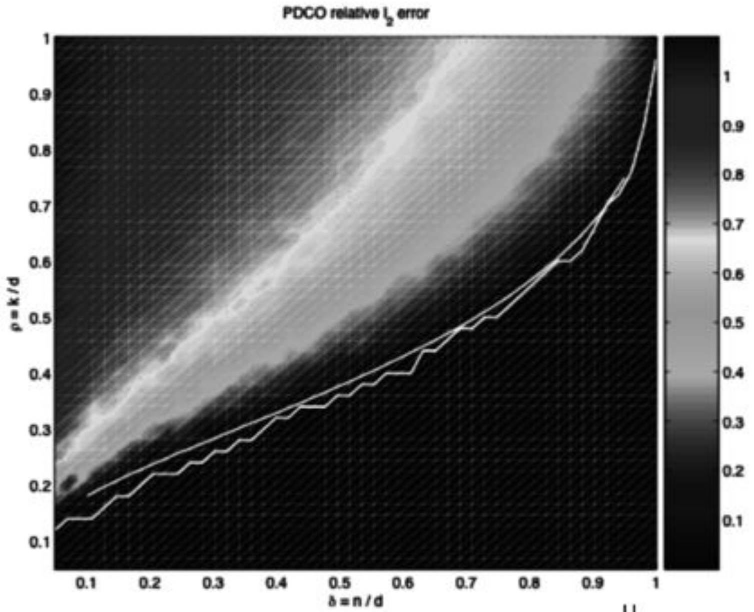
Sparsity

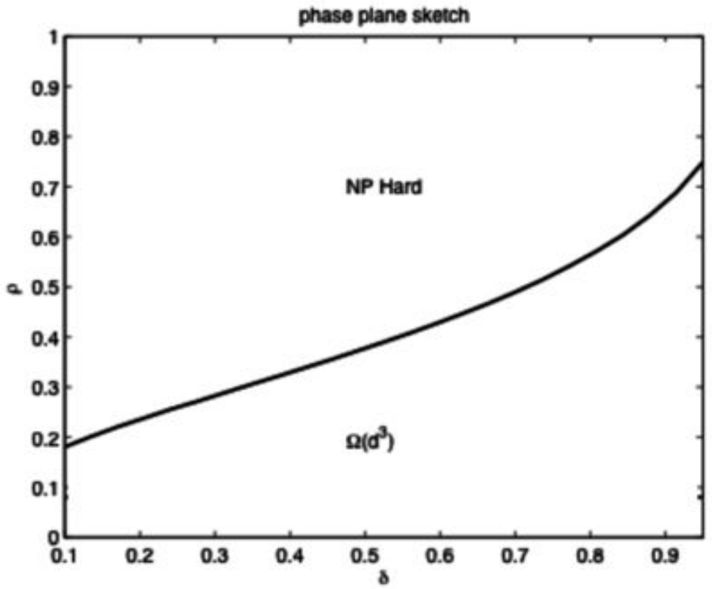
$$\min \|x\|_0 \text{ subject to } y = Ax.$$

$$\min \|x\|_1 \text{ subject to } y = Ax.$$

Sparsity Phase Transition

- y-axis: # non-zero coefficients / # examples
- x-axis: # of features / # of examples



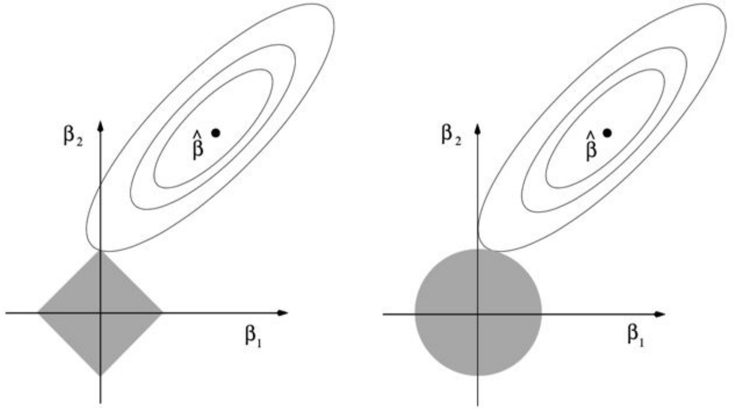


$$\|x - x_0\|_2 / \|x_0\|_2$$

Lasso

$$\min \|y - Ax\|_2^2 \text{ subject to } \|x\|_1 \leq q.$$

$$\min \|y - Ax\|_2^2/2 + \lambda\|x\|_1$$



NYU

Anomaly Detection

Anomaly Detection

- Unbalanced dataset: mostly positive examples, a few negatives
- Hard to generalize from negative examples, different anomaly types
- Supervised with different characteristics

Training Data

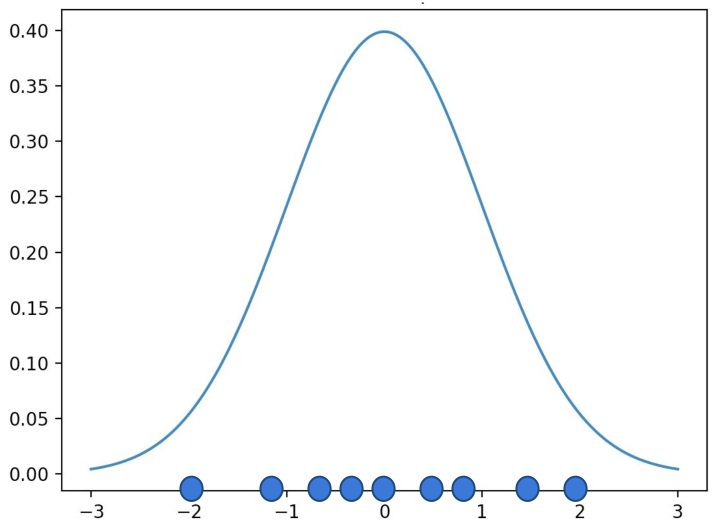
- Given a dataset $D = \{x^{(1)}, \ldots, x^{(m)}\}$ of vectors with labels $y_1, \ldots, y_n$
- Each example i is a vector of features $x^{(i)} = \left(x_1^{(i)}, \ldots, x_n^{(i)}\right)$
- Labels are binary, 0 for normal example, 1 for anomaly. Nearly all examples are labeled 0.

Gaussian Model

- Consider each feature j across all m examples: $x_j = \left(x_j^{(1)}, \ldots, x_j^{(m)}\right)$
- For many examples, feature can be estimated by normal distribution, or transformed to a normal distribution $\mathcal{N}(\mu_j, \sigma_j^2)$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}\left(\mu_j - x_j^{(i)}\right)^2$$



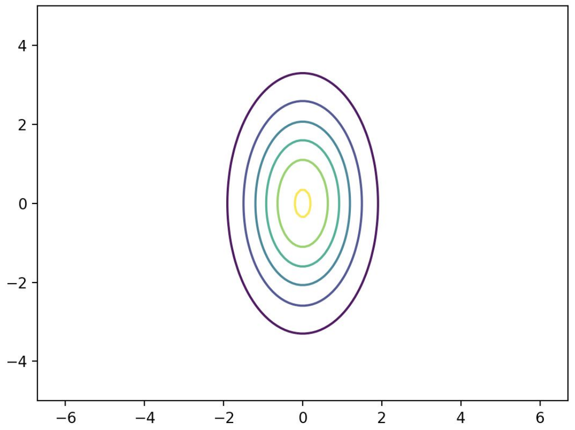
Independent Gaussians Model

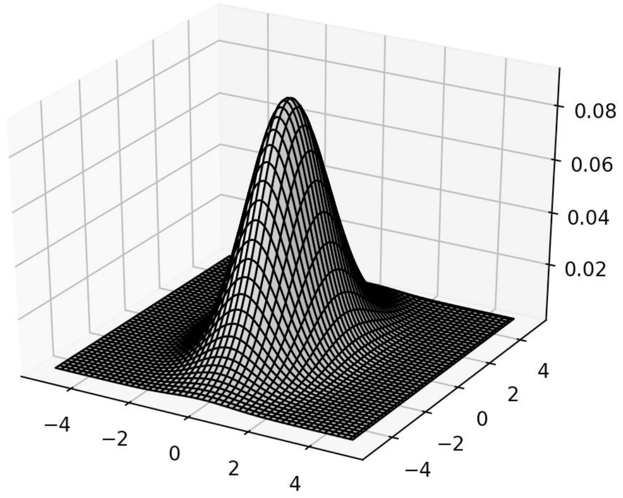
- Probability of new example *x* can be modeled as:

$$p(x) = \prod_{j=1}^{n} p(x_j | \mu_j, \sigma_j^2)$$

Anomaly Detection

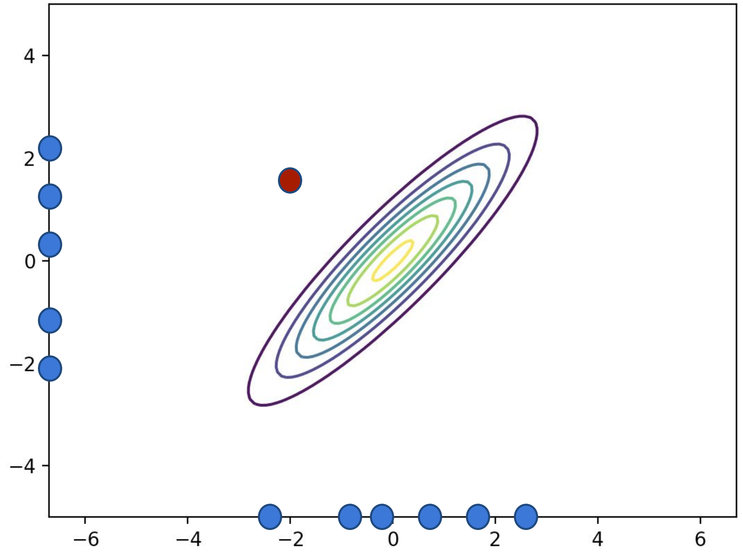
- Set threshold ε to small value
- Detect outliers example x by $p(x) < \varepsilon$
- An example in 2d, two features:





Problem with Independent Gaussians Model

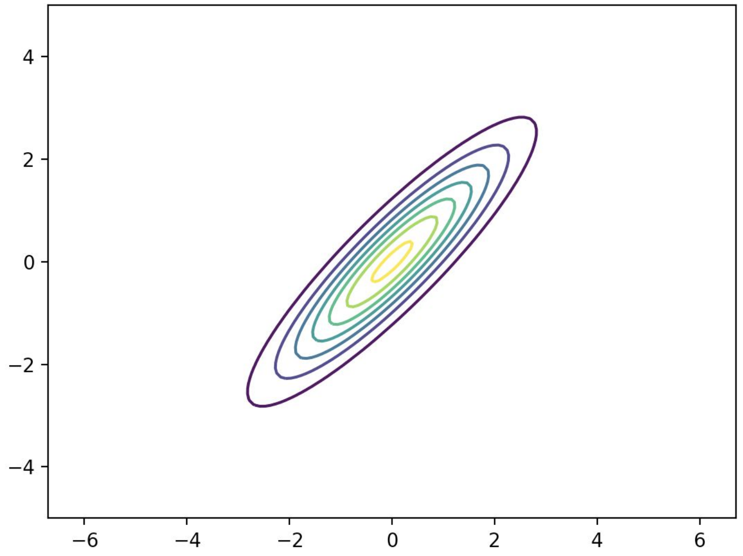
- Examples which fall within each Gaussian feature independently may be outliers



Solution: Multivariate Gaussian

- Use multivariate Gaussian model

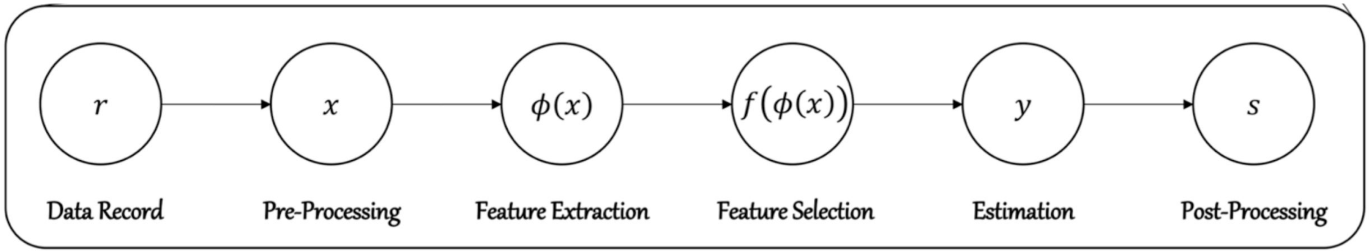
$$p(X = x) = \mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$



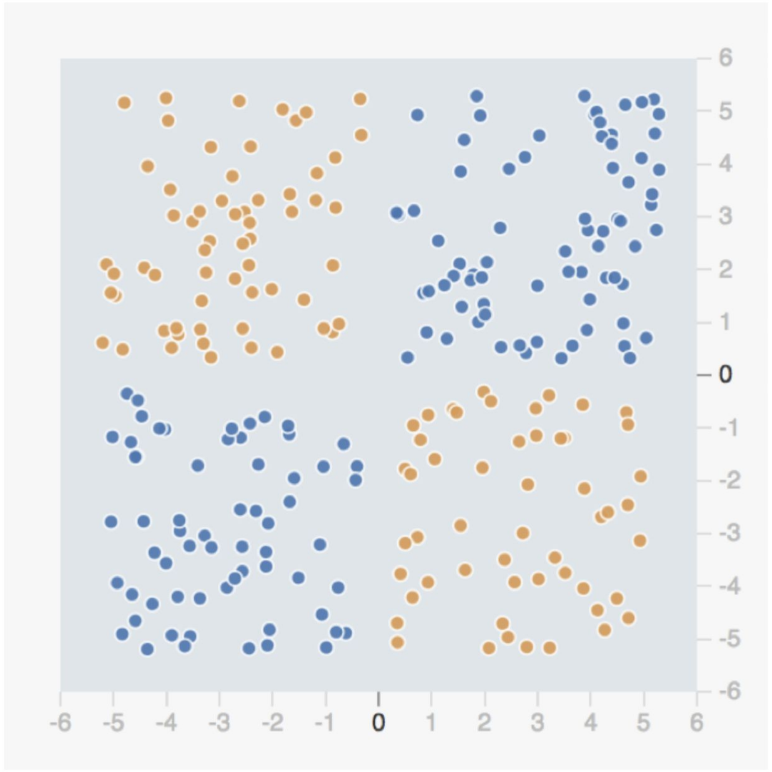
Feature Extraction

Feature Extraction

- Training data x
- Learning finds weights of model by optimization
- Feature extraction finds $\phi(x)$ by prior domain knowledge
- Feature extractor $\phi(x)$ should select features useful for prediction
- Feature extractor defines a hypothesis class, model family of possible predictors
- Learning is choosing a predictor from that space based on data



Feature Extraction



$f_w(x) = w^T x \qquad (x_1 x_2)$

Random Forests

Random Forest

- Grow many deep regression trees to randomized versions of training data and average them.
- Simply averaging the same tree learning algorithm on different data subsets may yield correlated predictors.
- Learn trees based on random subsets of features as well as random subsets of data.

Random Forests

Given a training sample $d=(X_{nxp},y)$

Set number of variables to $m < p$, and number of trees to T.

For $t = 1,2,\dots,T$:

1. Create bootstrap version of training data d_t by randomly sampling the n rows with replacement n times.
2. Grow a maximal depth tree g_t using data in d_t sampling m of p features at random prior to making each split.
3. Save tree and bootstrap sampling frequencies for each example.

Compute random forest fit at any prediction point as the average of the trees

NYU

Boosting

Boosting

- Iteratively grow shallow trees to the residuals and build an adaptive model of sum of trees.

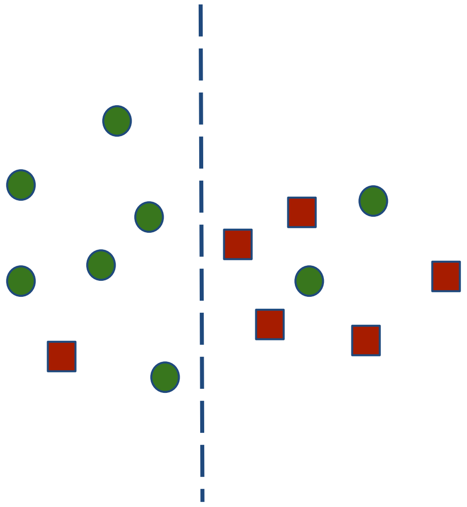
NYU

Adaboost

- Binary classification {-1,1}
- Iteratively fit classifiers to weighted training data
- Increase weight of misclassified examples
- Final classification by weighted majority vote

Adaboost

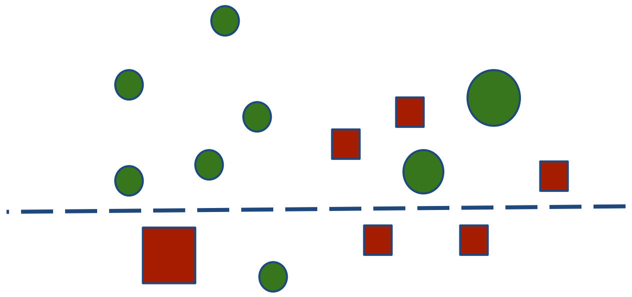
- Learn classifier on original data



$f_1(x)$

Adaboost

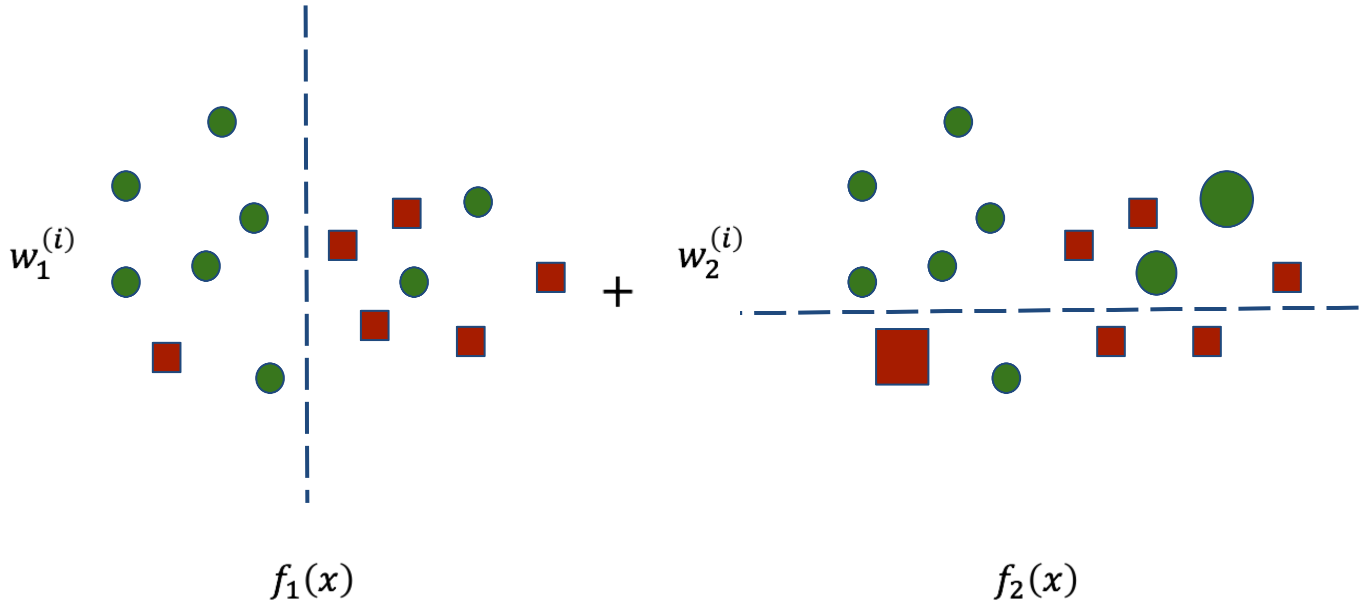
- Increase weight of misclassified examples
- Learn classifier on weighted data



$f_2(x)$

Adaboost

- Ensemble is weighted sum of classifiers



NYU

Adaboost

initialize weights $w_1^{(i)} = \frac{1}{m}$

fit classifier minimizing

$$J_t = \sum_{i=1}^{m} w_t^{(i)} I\left(f_t\left(x^{(i)}\right) \neq y^{(i)}\right)$$

compute misclassification error

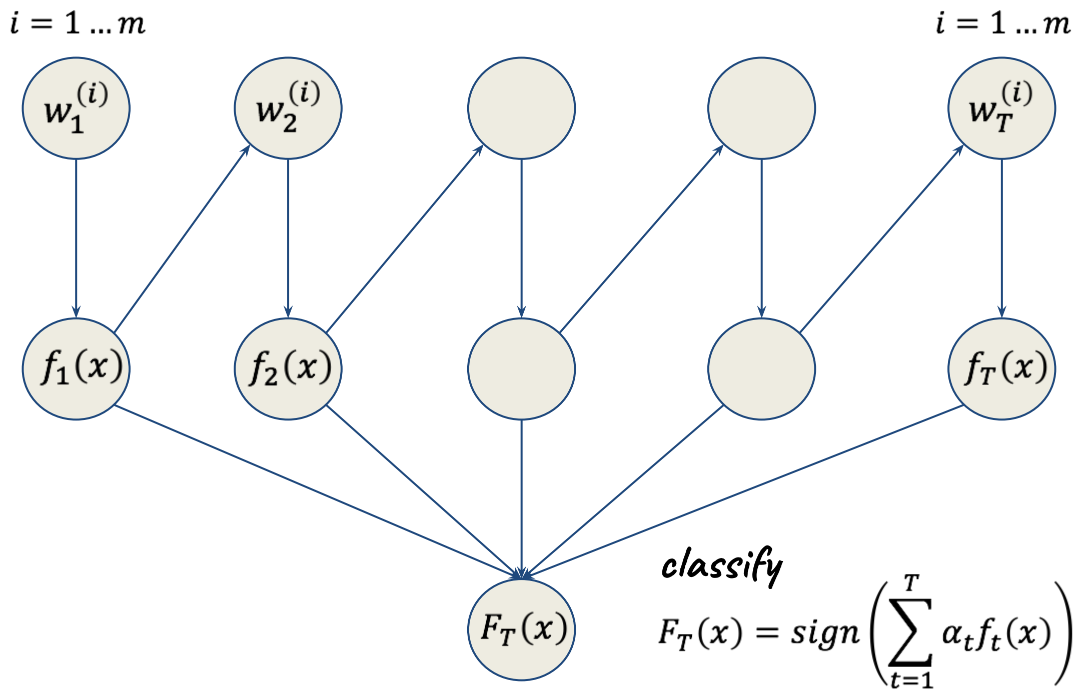
$$err_t = \frac{\sum_{i=1}^{m} w_t^{(i)} I\left(f_t\left(x^{(i)}\right) \neq y^{(i)}\right)}{\sum_{i=1}^{m} w^{(i)}}$$

evaluate

$$\alpha_t = \log\left(\frac{1 - err_t}{err_t}\right)$$

update

$$w_{t+1}^{(i)} = w_t^{(i)} e^{\left(\alpha_t I\left(f_t\left(x^{(i)}\right) \neq y^{(i)}\right)\right)}$$



classify

$$F_T(x) = sign\left(\sum_{t=1}^{T} \alpha_t f_t(x)\right)$$

Gradient Boosting

Gradient Boosting with Squared Error Loss

- Build model by iteratively fitting shallow regression tree (weak learner) to residuals.
- Shrink tree by learning rate and add to model.
- Each tree amends errors made by ensemble of previously grown trees.

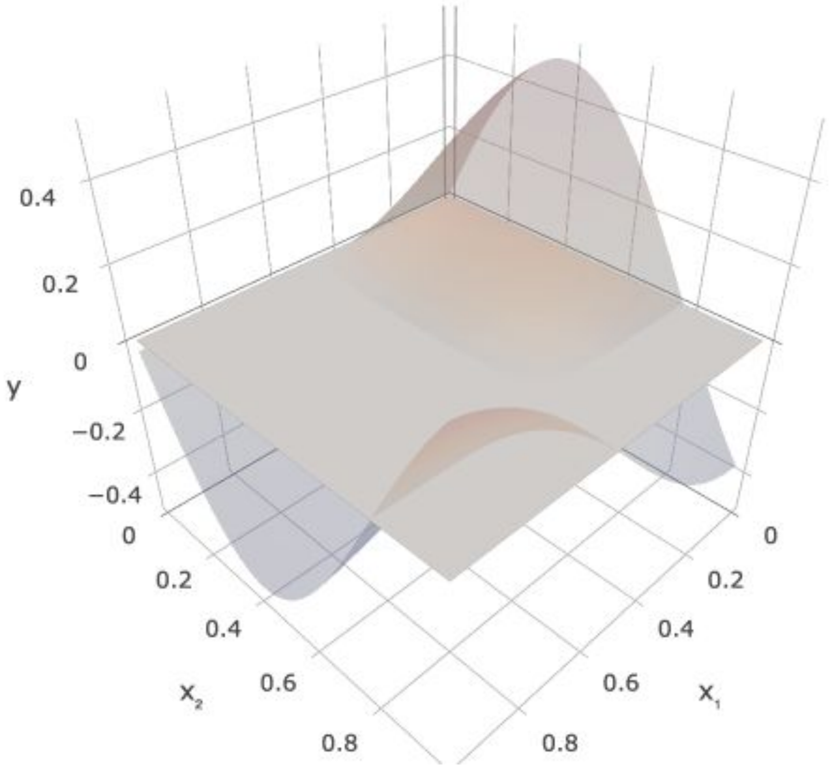
Gradient Boosting Parameters

- Tree depth *d*
- Number of iterations *t*
- Learning rate (shrinkage factor) *eps*

Gradient Boosting

depth 0

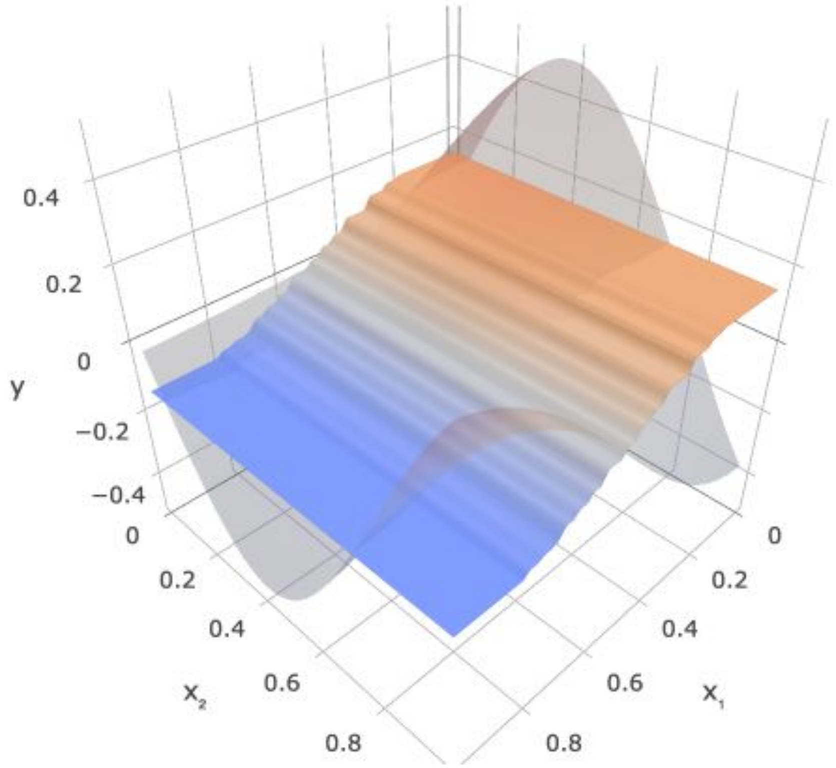
goal is to reconstruct f(x)



Gradient Boosting

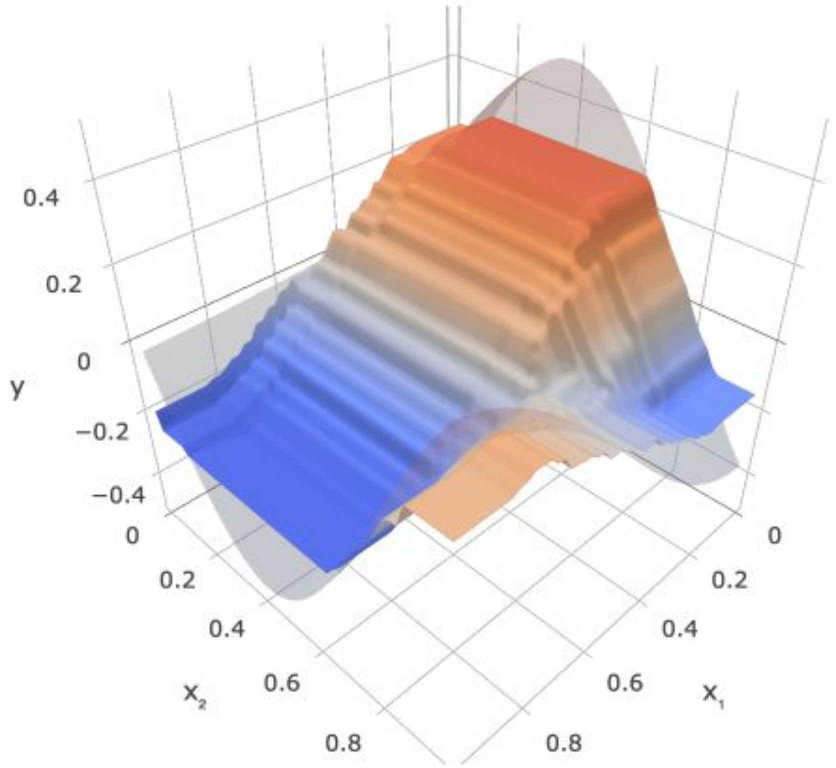
depth 1

$G_{100}(x)=g_1(x)+\ldots+g_{100}(x)$



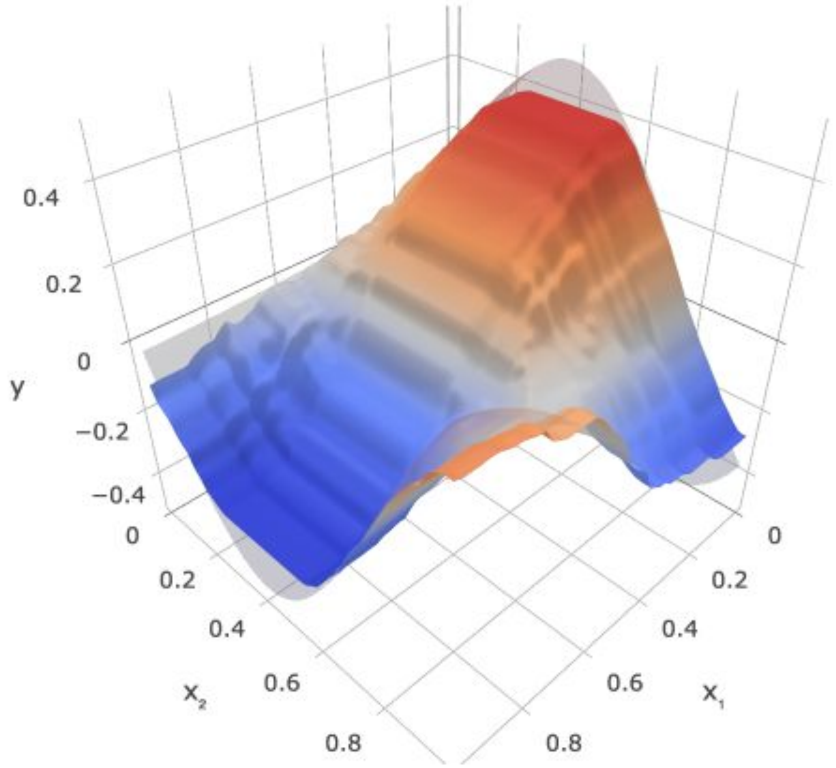
Gradient Boosting

depth 2



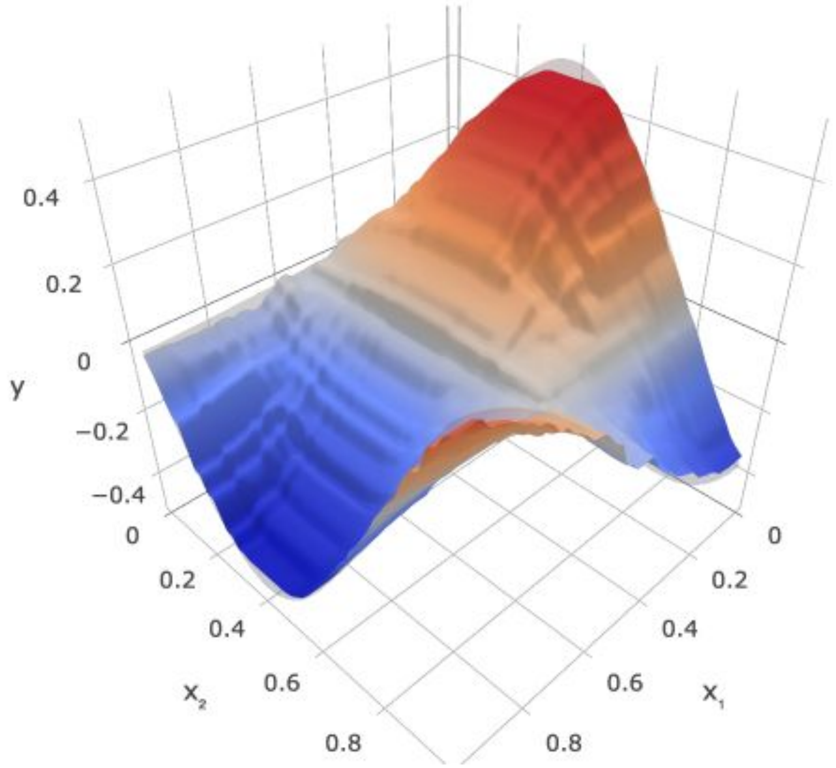
Gradient Boosting

depth 3



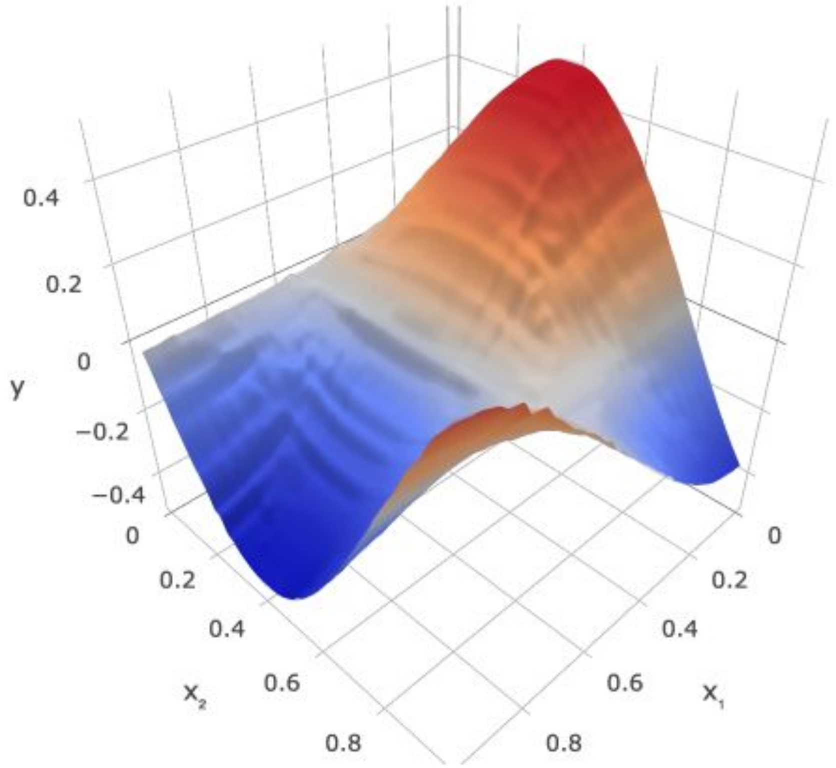
Gradient Boosting

depth 4



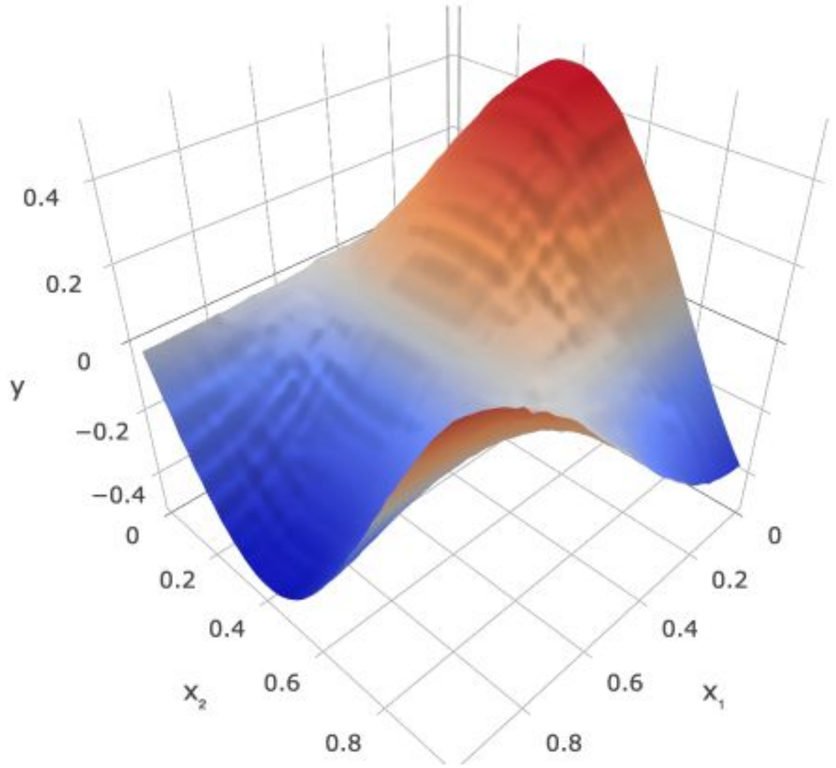
Gradient Boosting

depth 5



Gradient Boosting

depth 6



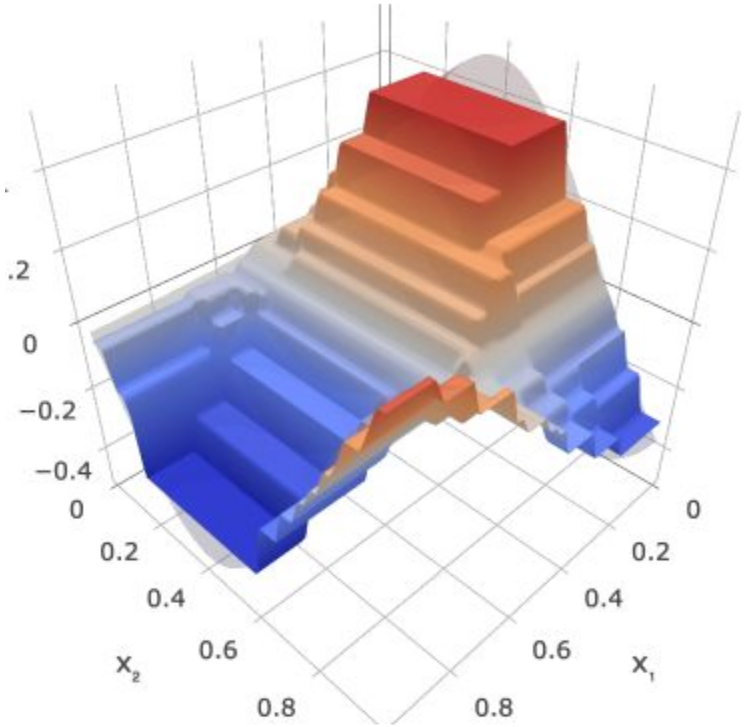
Gradient Boosting

At each iteration compute residual: $r(x) = f(x) - G_i(x)$.

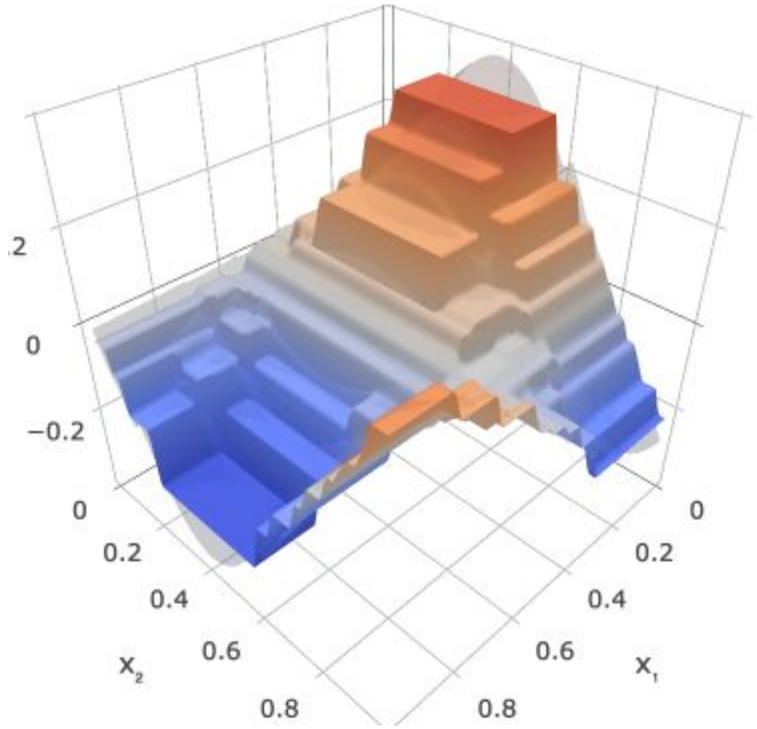
Train new tree $g(x)$ to reconstruct residual $r(x)$.

Add to approximation multiplied by learning rate.

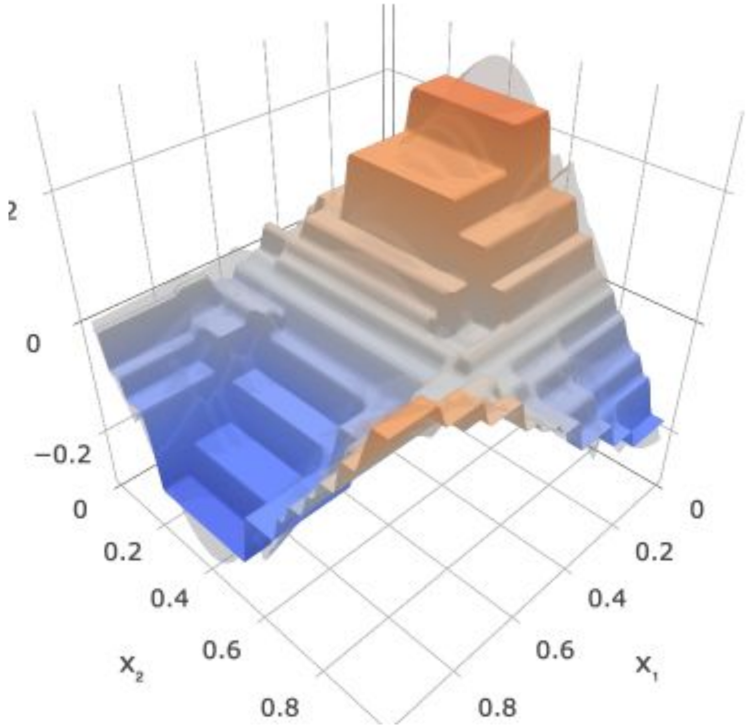
Residual



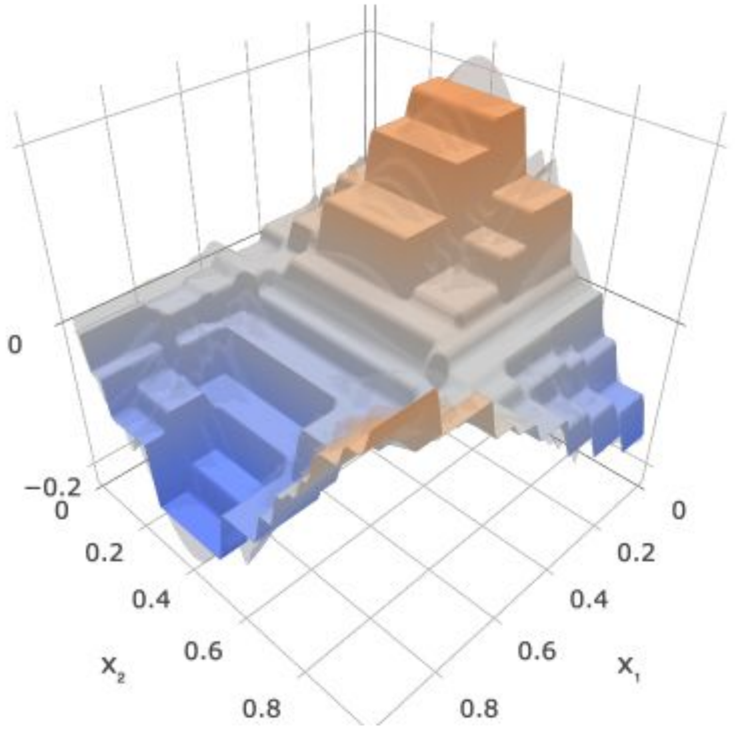
Residual



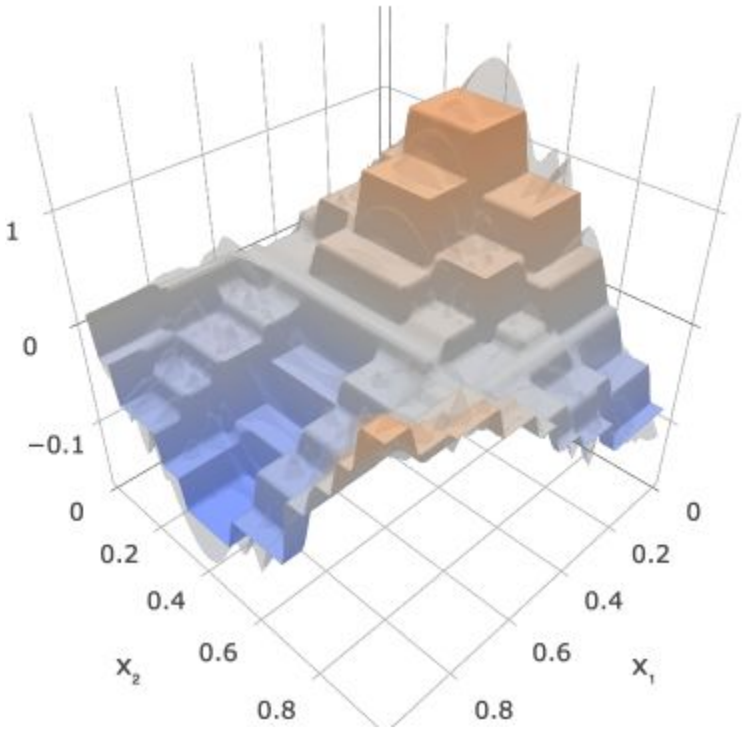
Residual



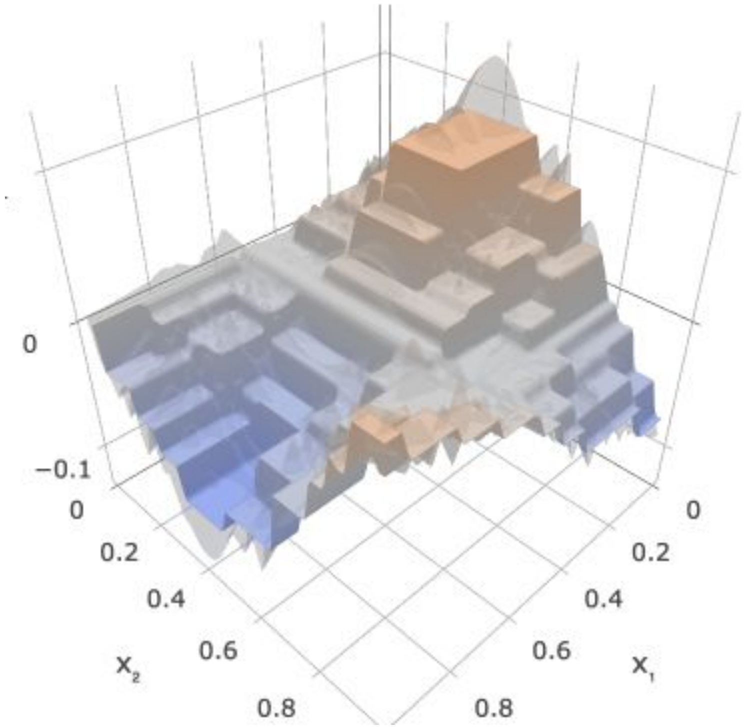
Residual



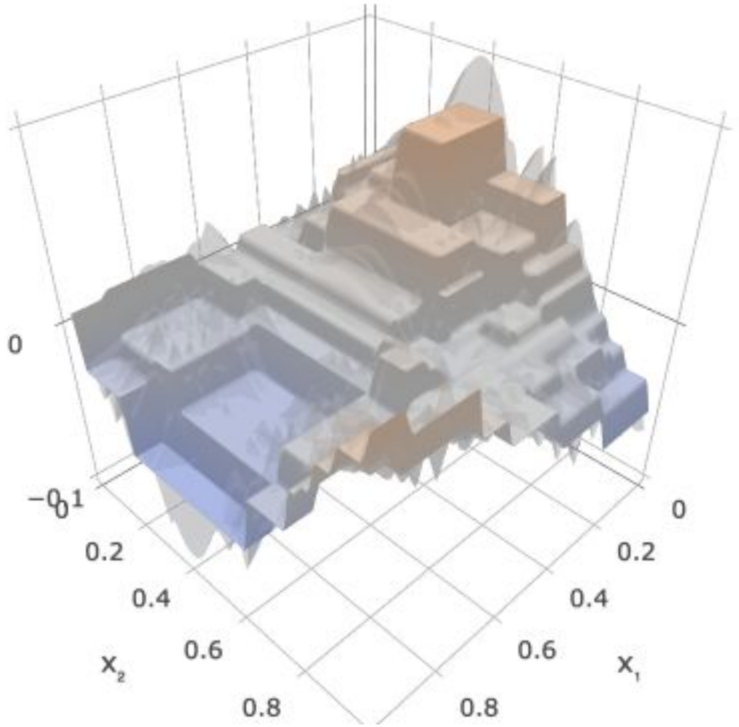
Residual



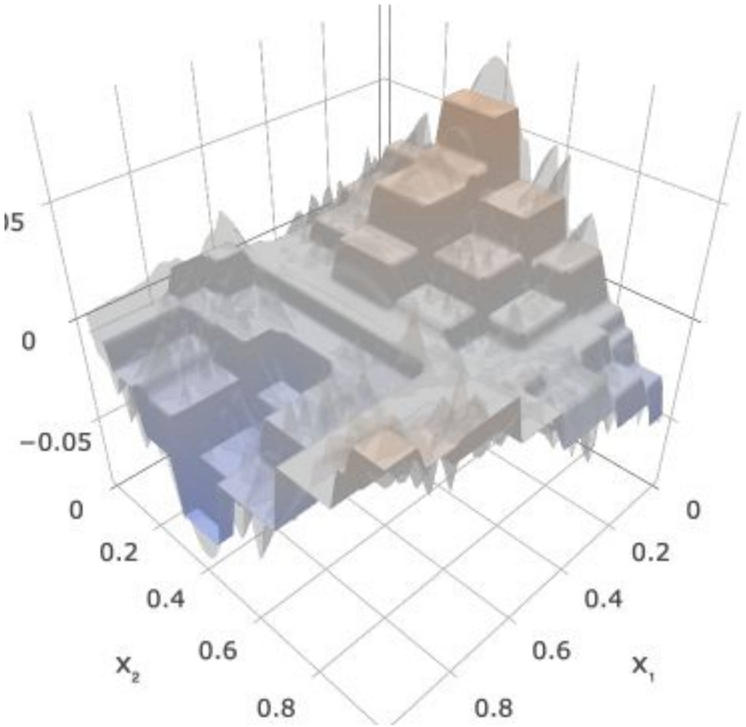
Residual



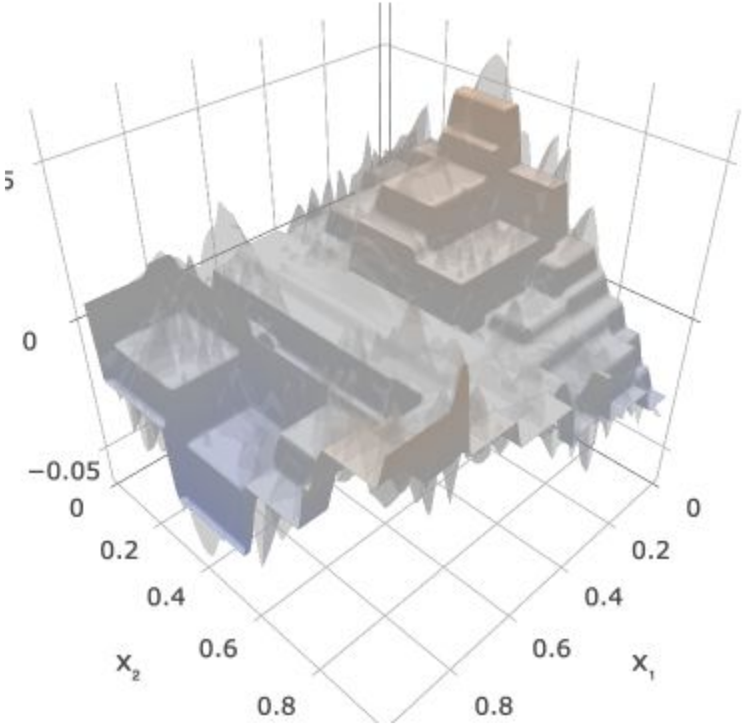
Residual



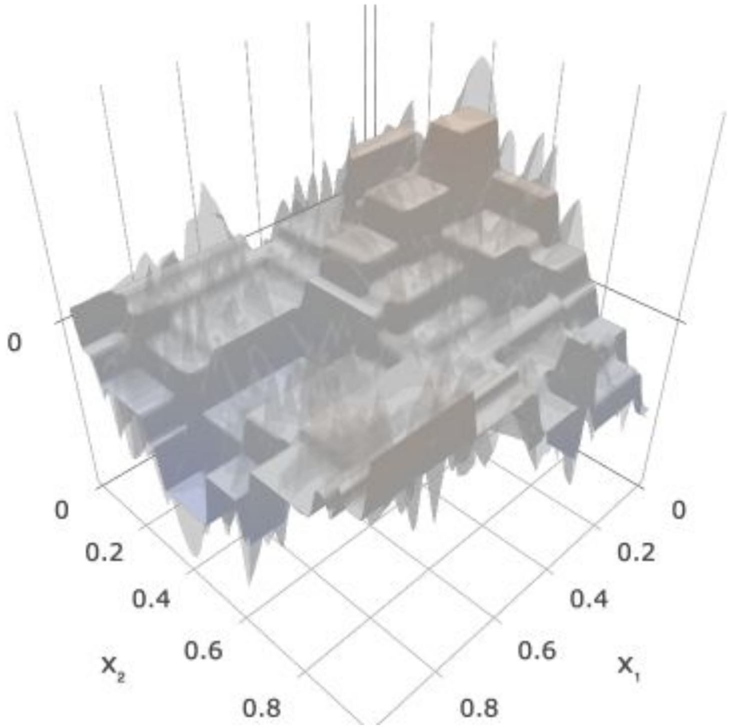
Residual



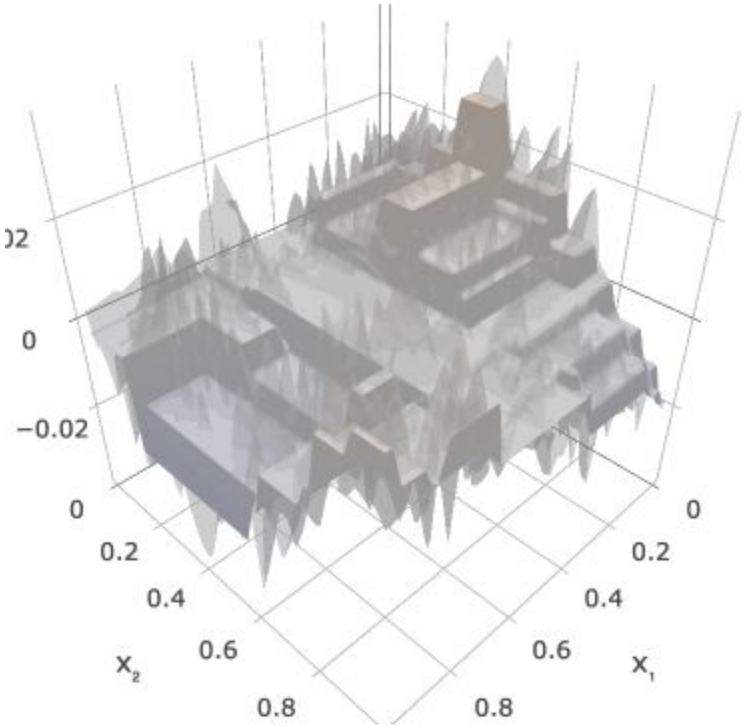
Residual



Residual



Residual



Gradient Boosting with Squared Error Loss

Given a training sample $d=(X,y)$

Set number of steps t, shrinkage factor eps, and tree depth d.

Set initial fit to $G_0 = 0$ and residual $r = y$.

For $t = 1,2,\ldots,T$:

1. Fit a regression tree g_t to data (X,r) grown best-first to depth d.
   Total number of splits are d. Each successive split is made to that terminal node which yields biggest reduction in residual sum of squares.
2. Update fitted model with shrunken version of g_t: $G_t = G_{t-1} + eps\ g_t$
3. Update residuals: $r_i = r_i - eps\ g_t(x_i)$ for $i=1,\ldots,n$

Return sequence of fitted functions $G_1, G_2, \ldots, G_T$