Linear Models and ML Fundamentals I

ML4SE

Denis Litvinov

September 7, 2021

Table of Contents

Prerequisites

- Linear Algebra
- Analysis
- Probability Theory
- Statistics
- Numeric Optimization

Course Content

1. Linear models, ML fundamentals
2. Distributed representations
3. Feed Forward Neural Networks
4. CNN, RNN
5. Language Modeling
6. Transformers
7. Transfer learning
8. Information Retrieval
9. Modern Transformer architectures

Evaluation

- HW1 - Assumptions of classical machine learning models
- HW2 - NN for classification
- HW3 - Language Modeling
- HW4 - Seq2seq
- HW5 - Metric Learning
- report
- activity

$$\begin{aligned} Total = round(&0.12 * HW1 + 0.12 * HW2 + 0.12 * HW3 \\ &+ 0.12 * HW4 + 0.12 * HW5 \\ &+ 0.2 * report + 0.2 * activity) \end{aligned}$$

Literature I

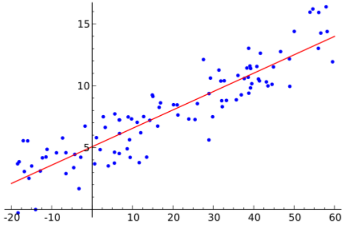
- Goodfellow I. Deep Learning. MIT Press, 2016
- Bishop C.M. Pattern Recognition and Machine Learning. Springer, 2006
- Hastie, T. The elements of statistical learning : data mining, inference, and prediction. Springer, 2009
- Sheldon A. Linear Algebra Done Right. Springer, 3d edition, 2015
- Schapire. Boosting: Foundation and Algorithms. MIT Press, 2012

Taxonomy

- Supervised vs Unsupervised vs Reinforcement Learning
- Supervised Learning: Classification vs Regression
- Discriminative vs Generative Model
- Structured vs Unstructured Prediction

Task Formulation

Given dataset $\{(x_i, y_i)\}_{i=1}^{N}$ of i.i.d. objects
Or, equvalently given:
$X \in R^{NxD}$ - feature matrix, where d is dimension of feature space and N - number of objects.
$Y \in R^{N}$ - target vector.
We want to find such algorithm $h \in H$ that $h(x) = \hat{y}$ "assigns for each object the right target value".



Loss Functions

$Loss : RxR \to R$ - loss function, that evaluates how bad our prediction for particular object are.
Some loss functions:

Mean Squared Error

$$Loss(\hat{y}, y) = (\hat{y} - y)^2$$

Mean Absolute Error

$$Loss(\hat{y}, y) = |\hat{y} - y|$$

Mean Absolute Percentage Error

$$Loss(\hat{y}, y) = \frac{|\hat{y} - y|}{y}$$

Quality Metrics

Coefficient of Determination

$$R^2(\hat{y}, y) = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \frac{1}{N}\sum_j y_j)^2}$$

Root Mean Squared Error

$$RMSE(\hat{y}, y) = \sqrt{\frac{1}{N}\sum_i (\hat{y}_i - y_i)^2}$$

Mean Absolute Error

$$MAE(\hat{y}, y) = \sqrt{\frac{1}{N}\sum_i |\hat{y}_i - y_i|}$$

Linear Regression

Here we explore linear regression with MSE loss

$$L_{MSE}(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y-\hat{y})^2$$

Usually we write

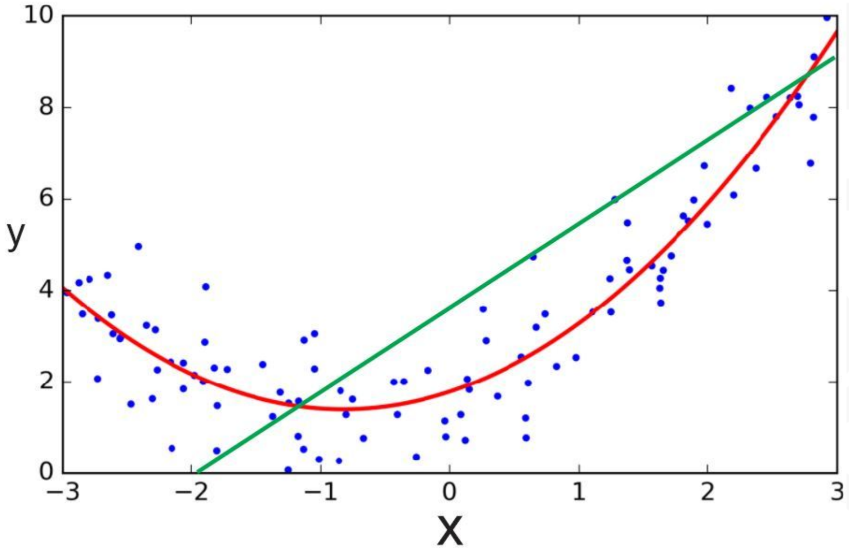
$$\hat{y} = w^T x + b$$

where w, b - model weights, and b is called *intercept*
For future convenience add b into vector x

$$(x)^T \to (1, x)^T$$

$$\hat{y} = w^T x$$

Polynomial Regression 1



Polynomial Regression 2

What if we can't approximate data with linear model? Use polynomial regression

$$\phi_n(x) = (1, x, x^2, x^3, .., x^n)^T$$

For example,

$$\phi_2(1, x_1, x_2) \rightarrow (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$$

And this way can still use linear model

$$\hat{y} = w^T \phi_n(x)$$

However, need to choose the right polynomial power.

Analytic solution of MSE linear regression 1

$$\hat{y} = Xw$$

$$\begin{aligned}
L_{MSE}(y, X) &= \frac{1}{N}(y - Xw)^T(y - Xw) \\
&= \frac{1}{N}(y^T - (Xw)^T)(y - Xw) \\
&= \frac{1}{N}(y^T y - y^T(Xw) - (Xw)^T y + (Xw)^T(Xw)) \\
&= \frac{1}{N}(y^T y - 2y^T(Xw) + (Xw)^T(Xw))
\end{aligned}$$

Analytic solution of MSE linear regression 2

$$\frac{\partial(BA)}{\partial A} = B^T$$
$$\frac{\partial(A^T B)}{\partial A} = B$$
$$\frac{\partial(A^T A)}{\partial A} = 2A$$
$$\frac{\partial(A^T BA)}{\partial A} = AB + A^T B$$

Analytic solution of MSE linear regression 3

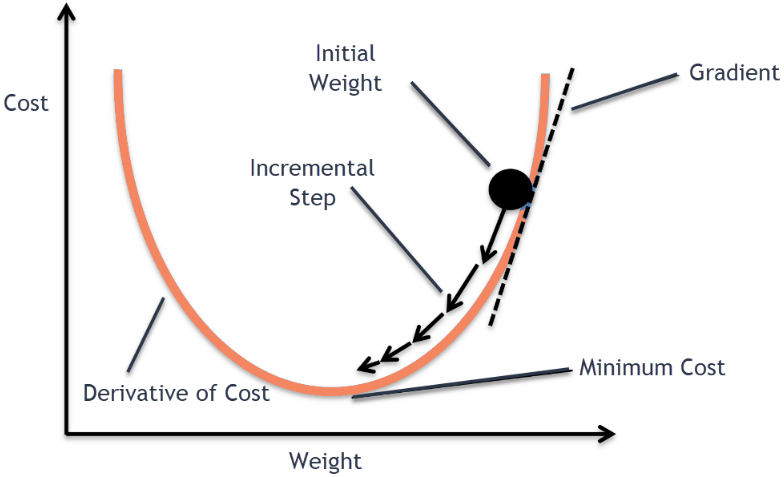
$$\frac{\partial L_{MSE}}{\partial w} = \frac{1}{N}(-2X^T y + 2X^T X w) = 0$$

$$X^T X w = X^T y$$

$$w = (X^T X)^{-1} X^T y$$

Properties of analytic solution depends on $(X^T X)^{-1}$
Remember, that if $det(A) \to 0$ then A^{-1} is numerically unstable

Iterative solution 1



Iterative solution 2

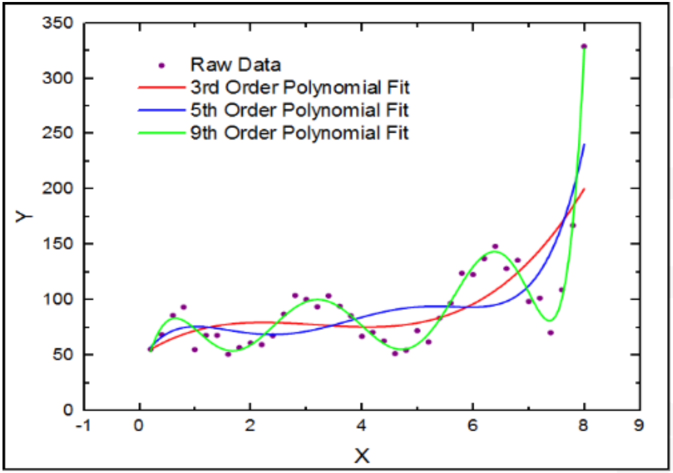
As we observed previously

$$\nabla_w L_{MSE} = \frac{1}{N} X^T (Xw - y)$$

1. $w^{(0)}$ = random init
2. at time t $w^{(t)} = w^{(t-1)} - \alpha \nabla_w L_{MSE}(w^{(t-1)})$
3. until convergence $||\nabla_w L_{MSE}(w^{(t-1)})|| < \epsilon$ OR $||w^{(t)} - w^{(t-1)}|| < \epsilon$ OR number of iterations exceeds predefined maximum

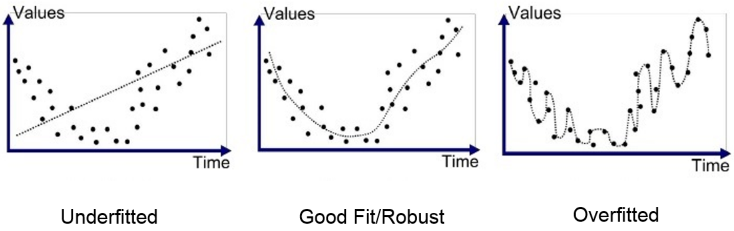
No issues of $(X^T X)^{-1}$ numeric stability!

What curve fits the data the best?



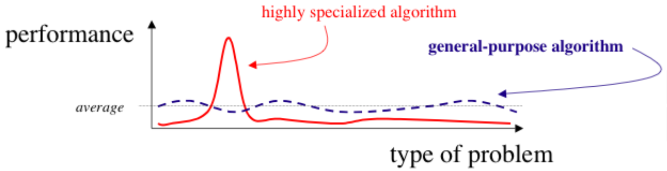
Overfitting

Overfitting is a situation, when a model fitted on a train dataset shows worse perfomance on a test dataset.
It corresponds to the fact, that model learns the given dataset but do not generalize to unseen data from the same distribution.
Every model does overfit!



No Free Lunch Theorem

The No Free Lunch Theorems state that any one algorithm that searches for an optimal cost or fitness solution is not universally superior to any other algorithm.
"If an algorithm performs better than random search on some class of problems then in must perform worse than random search on the remaining problems." (No Free Lunch Theorems for Optimisation)
How that affects machine learning? **Every machine learning algorithm explicitly or implicitly implies some assumptions made about observed data**. So by contradicting these assumptions for every algorithm we create such dataset, where it achieves bad performance.



Regularization

General Form

$$L_{reg}(y, X, w) = L(y, X) + \lambda R(w)$$

where
$R(w)$ - regularization term
λ - coef of regularization (regularization strength)
In linear modes we usually use L_p norm regularization:

$$R(w) = ||w||_p^p$$

For MSE with L_2 regularization

$$L_{MSE} = \frac{1}{N}||y - Xw||_2^2 + \frac{\lambda}{2}||w||_2^2$$

$$\nabla_w L_{MSE} = \frac{1}{N}X^T(Xw - y) + \lambda w$$

L_2 Regularization as stabilization of matrix inverse

From analytic solution we have

$$\nabla_w L_{MSE} = \frac{1}{N} X^T (Xw - y) + \lambda w = 0$$

Up to scaling factor λ

$$\nabla_w L_{MSE} = X^T (Xw - y) + \lambda w = 0$$

$$(X^T X + \lambda I) w = X^T y$$
$$w = (X^T X + \lambda I)^{-1} X^T y$$

Thus we have stabilization of matrix inverse

$$(X^T X)^{-1} \to (X^T X + \lambda I)^{-1}$$

Also called Tikhonov regularization.

L_2 as Gaussian prior on weights 1

Bayesian view.
We have samples $\{(x_i, y_i)\}_{i=1}^N$ from some distribution $P(x, y)$
Suppose

$$y_i = w^T x_i + \epsilon$$

, where

$$\epsilon \sim N(0, \sigma^2)$$

Thus we can construct likelihood function (remember Maximum Likelihood Estimation)

$$P(y_1, .., y_N | x_1, .., x_N) = \prod_{i=1}^N N(y_i | w^T x_i, \sigma^2)$$

where
$N(y_i | w^T x_i, \sigma^2)$ is a Gaussian distribution with mean $w^T x_i$ and variance σ^2

L_2 as Gaussian prior on weights 2

$$N(y_i|w^T x_i, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y_i - w^T x_i)^2}{2\sigma^2}}$$

Now imposing Gaussian prior on weights $w \sim N(0, \lambda^{-1})$

$$P(y_1, .., y_N | x_1, .., x_N) = \prod_{i=1}^{N} N(y_i|w^T x_i, \sigma^2) N(w|0, \lambda^{-1})$$

by MLE we would like to maximize

$$\log \prod_{i=1}^{N} N(y_i|w^T x_i, \sigma^2) N(w|0, \lambda^{-1}) \to \max_w$$

L_2 as Gaussian prior on weights 3

$$\sum_{i=1}^{N} \log N(y_i|w^T x_i, \sigma^2) + N * \log N(w|0, \lambda^{-1}) \to \max_w$$

$$-\sum_{i=1}^{N} \log N(y_i|w^T x_i, \sigma^2) - N * \log N(w|0, \lambda^{-1}) \to \min_w$$

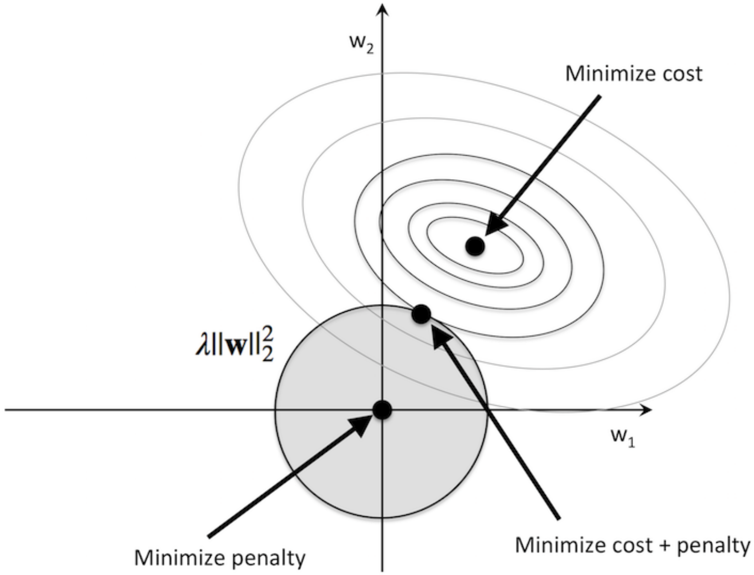
$$-\sum_{i=1}^{N} (-\frac{1}{2\sigma^2}(y_i - w^T x_i)^2) - N * (-\frac{1}{2\lambda^{-1}} w^T w) \to \min_w$$

$$\frac{1}{N\sigma^2} \sum_{i=1}^{N} (y_i - w^T x_i)^2 + \frac{\lambda}{2} w^T w \to \min_w$$

Up to scaling factor σ^2 we have familiar MSE loss with L_2 regularization

$$\frac{1}{N\sigma^2}(y - Xw)^T (y - Xw) + \frac{\lambda}{2}||w||_2^2 \to \min_w$$

L_2 as Gaussian prior on weights 4



L_2 regularization and early stopping 1

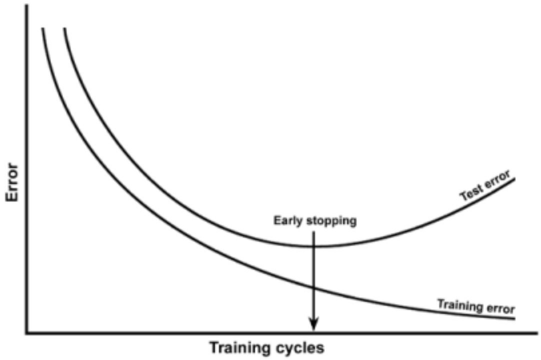
In Iterative solution we have mentioned some stopping criteria. We can imagine another one called *early stopping*:

1. split data into train and validation subsets
2. update model weights w on train dataset
3. keep track of the loss value on validation dataset
4. if on several consecutive iterations values of the loss function on validation dataset grows, than overfitting is observed $\to$ stop training

It can be shown, that number of consecutive iterations before early stopping τ can be expressed by coefficient of L_2 regularization λ

$$\tau \sim \frac{1}{\lambda}$$

L_2 regularization and early stopping 2



L_1 Regularization and sparsity 1

What if we use other norm for regularization?

$$R(w) = ||w||_1$$

For MSE with L_1 regularization

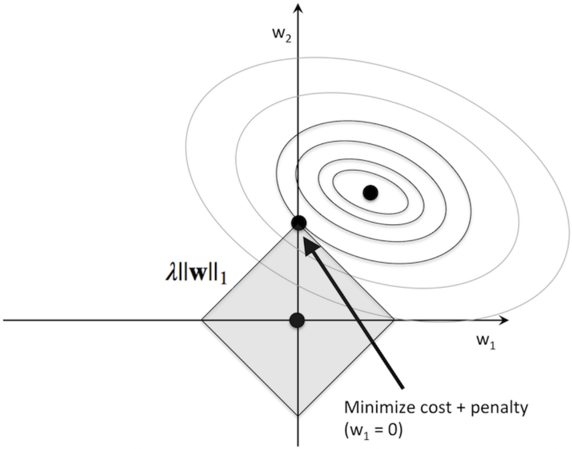
$$L_{MSE} = \frac{1}{N}||y - Xw||_2^2 + \frac{\lambda}{2}||w||_1$$

$$\nabla_w L_{MSE} = \frac{1}{N}X^T(Xw - y) + \lambda sign(w)$$

- L_1 norm is not differentiable at $w = 0$, but can be lower bounded by surrogate gradients (just say $\nabla_w R(0) \in [-1, 1]$)
- Gives sparse solutions: some of w components are 0
- Bayesian view on L_1 norm regularizer is a Laplacian prior on weights

$$P(x|\mu, b) = \frac{1}{2b}e^{-\frac{|x-\mu|}{b}}$$

L_1 Regularization and sparsity 2



L_1 Regularization and sparsity 3

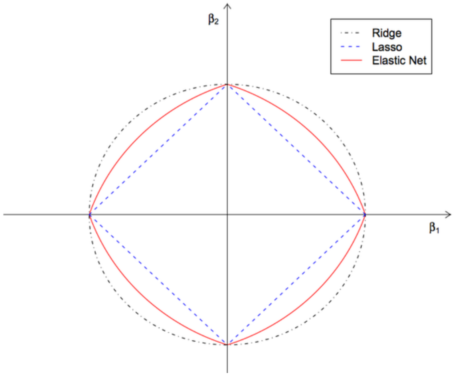
- L_1 can be used for feature selection
- **Remember, any L_p norm regularization shifts optimal solution w_***
- For linear models, if we want to make predictions with feature selection
  1. Train linear model with L_1 regularizer and select features with $|w_i| > 0$
  2. On selected subset of features, train linear model L_2 and use it for final prediction
- **Remember about situation with correlated features**

Elastic Net

$$L_{reg}(y, \hat{y}, w) = L(y, \hat{y}) + \lambda_1 ||w||_1 + \lambda_2 ||w||_2^2$$

Usually we would like to have convex combination in the form

$$L_{reg}(y, \hat{y}, w) = L(y, \hat{y}) + \lambda_1 ||w||_1 + (1 - \lambda_1) ||w||_2^2$$



Bias Variance Decomposition 1

Suppose our data is generated by:

$$y = f(x) + \epsilon$$

, where $\epsilon \in N(0, \sigma)$ is white noise.
We want to build such estimator, that:

$$\hat{y} = h(x)$$

is our prediction
Consider MSE regression

Bias Variance Decomposition 2

$$
\begin{aligned}
MSE &= E[(y - h(x))^2] \\
&= E[(y - f(x) + f(x) - h(x))^2] \\
&= E[(y - f(x))^2] + E[(f(x) - h(x))^2] - 2E[(y - f(x)(f(x) - h(x))] \\
&= E[\epsilon^2] + E[(f(x) - h(x))^2] - 2(E[yf(x)] - E[yh(x)] - E[f^2(x)] \\
&+ E[f(x)h(x)])
\end{aligned}
$$

Notes:

- since f is deterministic then $E[f^2(x)] = f^2(x)$
- since $E[y] = f(x)$ then $E[yf(x)] = f^2(x)$
- $E[yh(x)] = E[f(x)h(x)] + E[\epsilon h(x)] = E[f(x)h(x)] + 0$

Bias Variance Decomposition 3

$$
\begin{aligned}
MSE &= E[\epsilon^2] + E[(f(x) - h(x))^2] - 2(f^2(x) - E[f(x)h(x)] \\
&+ 0 - f^2(x) + E[f(x)h(x)]) \\
&= E[\epsilon^2] + E[(f(x) - h(x))^2] \\
&= E[\epsilon^2] + E[(f(x) - E[h(x)] + E[h(x)] - h(x))^2] \\
&= E[\epsilon^2] + E[(f(x) - E[h(x)])^2] + E[(E[h(x)] - h(x))^2] \\
&+ 2E[(E[h(x)] - h(x))(f(x) - E(h(x))] \\
&= E[\epsilon^2] + E[(f(x) - E[h(x)])^2] + E[(E[h(x)] - h(x))^2] \\
&+ 2(E[f(x)E[h(x)]] - E[E[h(x)]^2] - E[h(x)f(x)] + E[h(x)E[h(x)]])
\end{aligned}
$$

Notes:

- $E[fE[h(x)]] = f(x)E[h(x)]$
- $E[E[h(x)]^2] = E[h(x)]^2$
- $E[f(x)h(x)] = f(x)E[h(x)]$
- $E[h(x)E[h(x)]] = E[h(x)]^2$

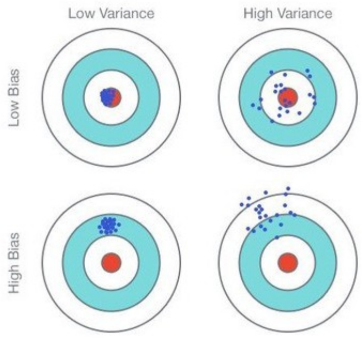
Bias Variance Decomposition 4

$$\begin{aligned} MSE &= E[\epsilon^2] + E[(f(x) - E[h(x)])^2] + E[(E[h(x)] - h(x))^2] \\ &\quad + 2(f(x)E[h(x)] - E[h(x)]^2 - f(x)E[h(x)] + E[h(x)]^2) \\ &= E[\epsilon^2] + E[(f(x) - E[h(x)])^2] + E[(E[h(x)] - h(x))^2] \\ &= Var[\epsilon] + E[(f(x) - E[h(x)])^2] + Var[h(x)] \\ &= Var[\epsilon] + bias^2 + Var[h(x)] \end{aligned}$$

Bias Variance Decomposition 5

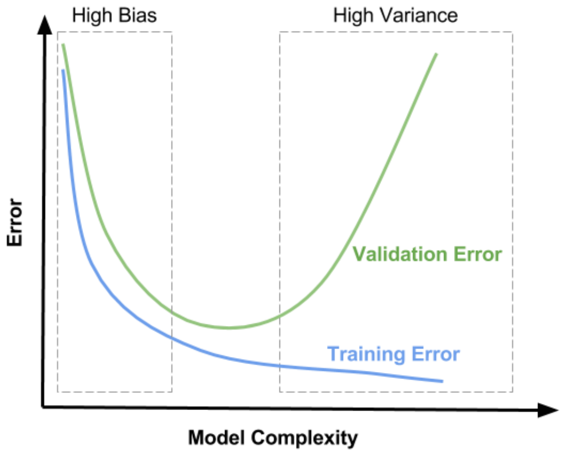
So prediction error can be decomposed into:

- variance of the noise
- bias of prediction
- variance of prediction



Validation curve 1

Validation curve is a dependence of model performance on the model complexity



Validation curve 2

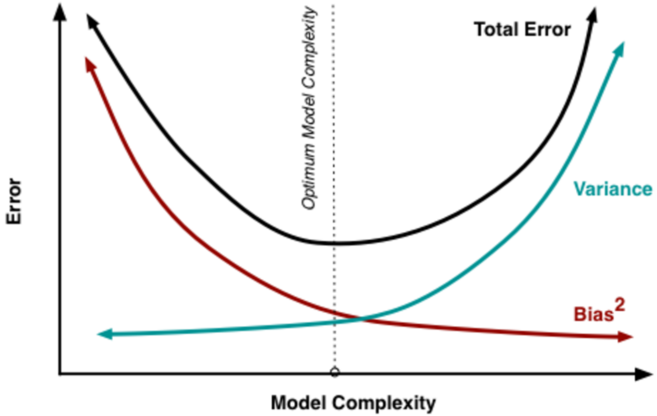


Figure: Generalization error

Learning curve 1

Learning curve is a dependence of model performance on the size of training dataset.

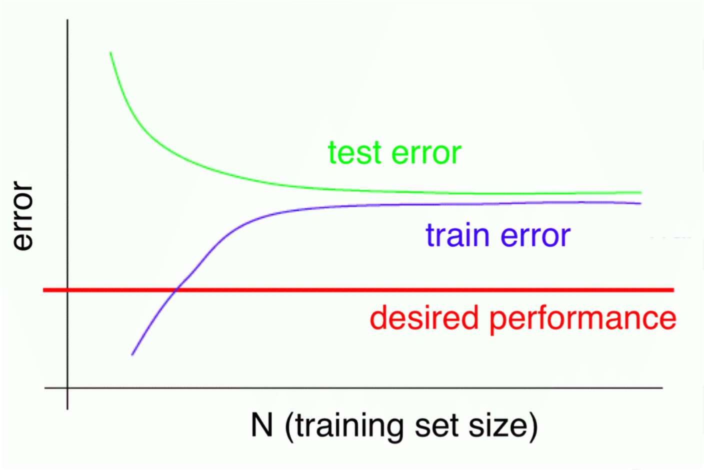


Figure: High bias

Learning curve 2

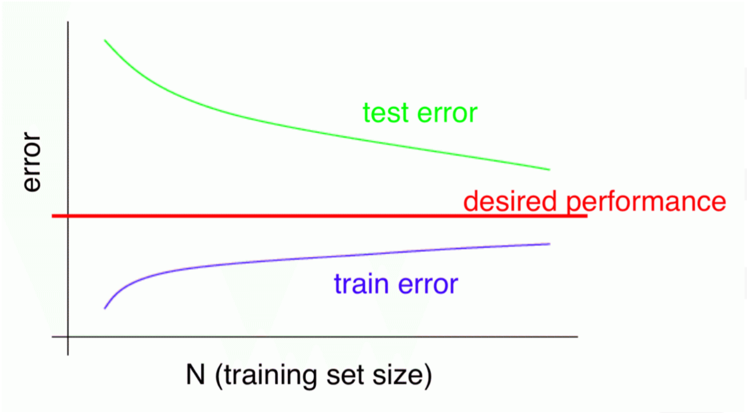


Figure: High variance

Decision Tree

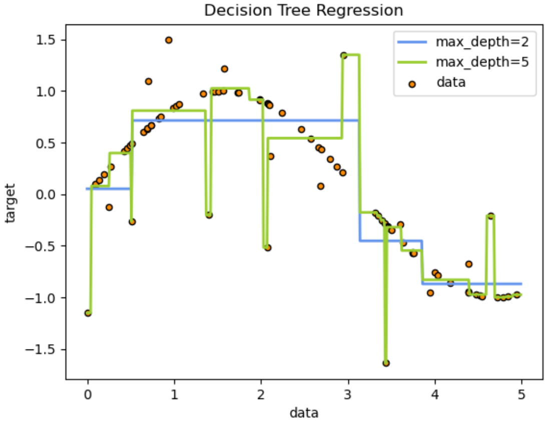


Figure: High variance

Definitions

Splitting criteria on vertex v: *j*-th component of feature vector *x* is less than the threshold *t*

$$\beta_v(x; j, t) = [x_j < t]$$

Greedy algorithm to build decision tree
Given a vertex *v*
for every feature *f*:
for every threshold *t* on f:
estimate chosen splitting criterion
Select *t* and *f* that maximizes that criterion. Make a split of incoming dataset into left *L* and right *R* subsets.

Impurity Criteria

Impurity criterion for dataset *R*: minimize loss function *Loss* with constant prediction *c*

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} Loss(y_i, c)$$

Impurity criterion for MSE regression

$$H(R) = \min_{c \in Y} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2$$

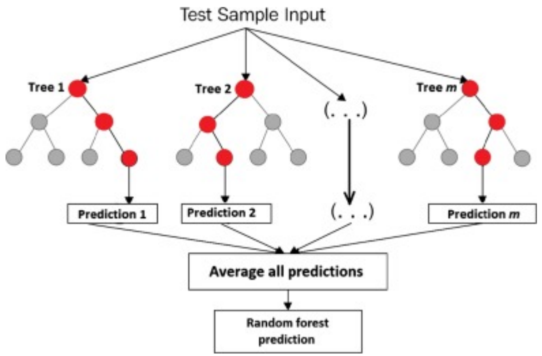
We know that for MSE task optimal value of *c* is

$$c_* = \frac{1}{|R|} \sum_{(x_j, y_j) \in R} y_j$$

Thus, impurity criterion is a variance of *y*

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c_*)^2 = Var[y]$$

Bagging. Random Forest 1



Bagging. Random Forest 2

1. Bootstrap = sampled subset with repetitions from initial dataset
2. Bagging = averaging over predictions of T base models trained on bootstraped datasets
3. Bagging gives $\frac{1}{T}$ factor in variance reduction under the assumption that models are not correlated

$$F(x) = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$$

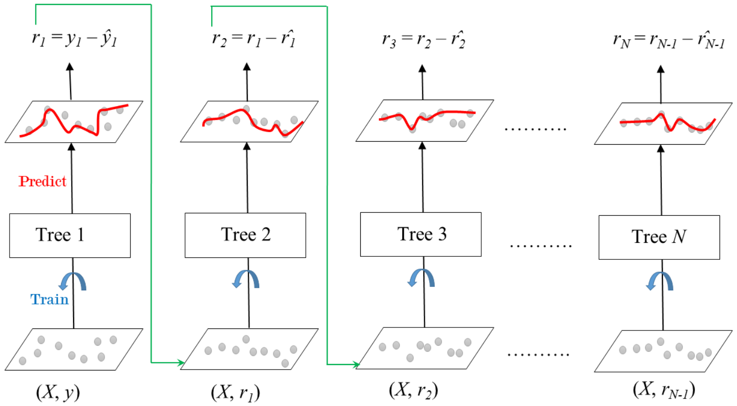
where
$F(x)$ - bagging ensemble model
$h_t(x)$ - base model, i.e. decision tree - must be uncorrelated
T - number of base models

Random Forest

1. Train N non-correlated decision trees in parallel
2. Average their predictions

Random forest is a bagging of decision trees with subsampling over features.

Gradient Boosting



Boosting

Any boosting can be described by

$$F(x) = \frac{1}{T}\sum_{t=1}^{T} w_t h_t(x)$$

where $F(x)$ - boosting ensemble model
$h_t(x)$ - base model, i.e. decision tree
w_t - weight coefficient for t base model
T - number of base models
Unlike bagging, boosting is aimed to reduce bias in the predictions.

Gradient Boosting

Iteratively build decision trees, trying to predict the gradient of error of the last tree.
Given finite set $(x_i, y_i)_{i=1}^N$ and some differentiable loss function $L(y, \hat{y})$.
1. select $h^{(0)} = const = \arg\min_{c=const} L(y, c)$
2. for t in $1..T$
2.1 pseudo-residuals $r_i^{(t)} = -\frac{dL(y, F^{(t-1)}(x_i))}{dy_i}$
On each iteration we wish to predict gradient of loss function over samples.
2.2 train $h^{(t)}$ to the dataset $\{(x_i, r_i^{(t)})\}_{i=1}^N$
2.3 $\alpha^{(t)} = \arg\min_\alpha \sum_i L(y_i, F^{(t-1)}(x_i) + \alpha h^{(t)}(x_i))$
2.4 $F^{(t)}(x) = F^{(t-1)}(x) + \alpha^{(t)} h^{(t)}(x)$
Note, that gradient has dimension equal to number of samples. =>
More data you have - more time to compute full gradient.