Machine Learning
Lecture 2: Linear Models

Radoslav Neychev

Outline

1. Previous lecture recap
2. Linear models overview
3. Linear Regression under the hood
4. Gauss-Markov theorem
5. Regularization in Linear regression
6. Model validation and evaluation
7. Gradient descent recap

Previous lecture recap

- Dataset, observation, feature, design matrix, target
- i.i.d. property
- Model, prediction, loss/quality function
- Parameter, Hyperparameter

Linear Models

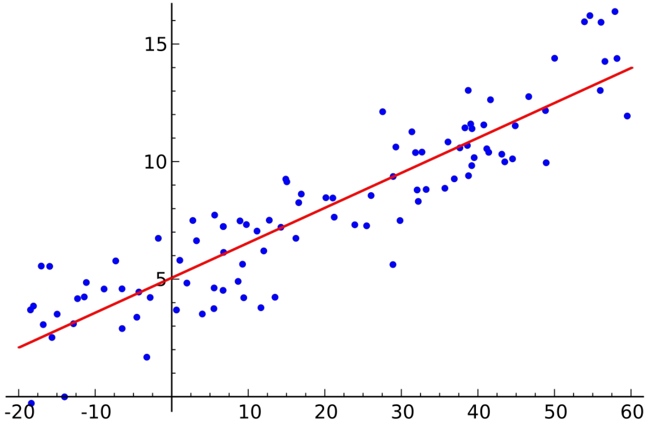
Linear models

$$\underset{\text{Dependent Variable}}{Y} = \underbrace{X_1 + X_2 + X_3}$$

Dependent Variable	Independent Variable
Outcome Variable	Predictor Variable
Response Variable	Explanatory Variable

Linear models

- Regression models



Estimated (or predicted) Y value for observation i

Estimate of the regression intercept

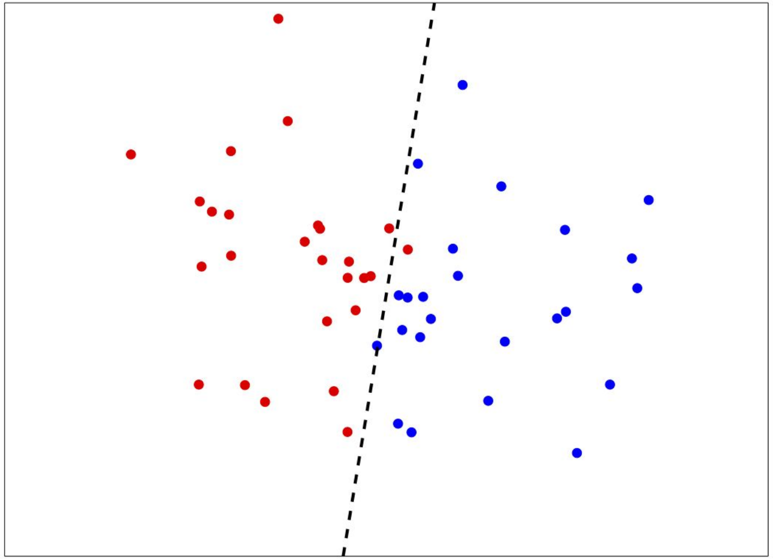
Estimate of the regression slope

Value of X for observation i

$$\hat{Y}_i = b_0 + b_1 X_i$$

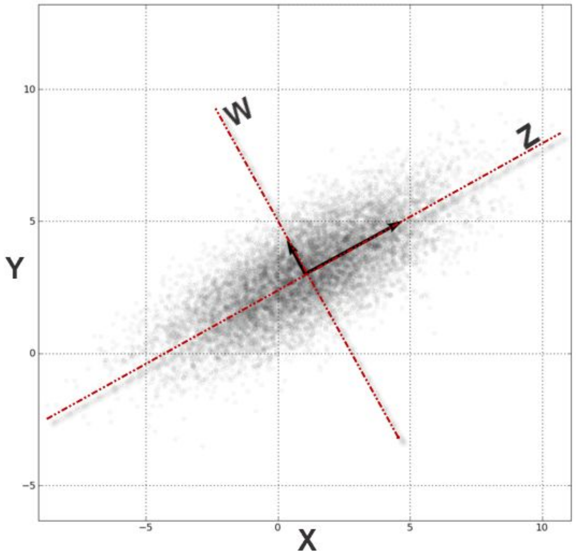
Linear models

- Regression models
- Classification models



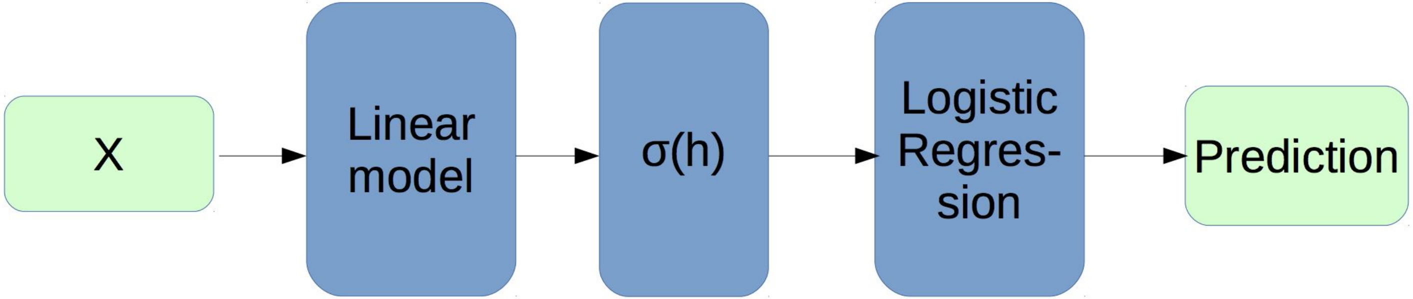
Linear models

- Regression models
- Classification models
- Unsupervised models (e.g. PCA analysis):



Linear models

- Regression models
- Classification models
- Unsupervised models (e.g. PCA analysis):
- Building block of other models (ensembles, NNs, etc.):



Actually, it's a neural network. We will meet it later.

Linear Regression

Linear regression problem statement:

- Dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \mathbb{R}$.

Linear regression problem statement:

- Dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \mathbb{R}$.
- The model is linear:

$$\hat{y} = w_0 + \sum_{k=1}^{p} x_k \cdot w_k = //\mathbf{x} = [1, x_1, x_2, \ldots, x_p]// = \mathbf{x}^T \mathbf{w}$$

Linear regression problem statement:

- Dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \mathbb{R}$.
- The model is linear:

$$\hat{y} = w_0 + \sum_{k=1}^{p} x_k \cdot w_k = //\mathbf{x} = [1, x_1, x_2, \ldots, x_p]// = \mathbf{x}^T \mathbf{w}$$

where $\mathbf{w} = (w_0, w_1, \ldots, w_n)$, w_0 is bias term

Linear regression problem statement:

- Dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \mathbb{R}$.
- The model is linear:

$$\hat{y} = w_0 + \sum_{k=1}^{p} x_k \cdot w_k = //\mathbf{x} = [1, x_1, x_2, \ldots, x_p]// = \mathbf{x}^T \mathbf{w}$$

where $\mathbf{w} = (w_0, w_1, \ldots, w_n)$, w_0 is bias term

we added an additional column of 1's to the design matrix to simplify the formulas

Linear regression problem statement:

- Dataset $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n, \quad y_i \in \mathbb{R}$.
- The model is linear:

$$\hat{y} = w_0 + \sum_{k=1}^{p} x_k \cdot w_k = //\mathbf{x} = [1, x_1, x_2, \ldots, x_p]// = \mathbf{x}^T \mathbf{w}$$

  where $\mathbf{w} = (w_0, w_1, \ldots, w_n)$, w_0 is bias term.

- Least squares method (MSE minimization) provides a solution:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \|Y - \hat{Y}\|_2^2 = \arg\min_{\mathbf{w}} \|Y - X\mathbf{w}\|_2^2$$

Analytical solution

Denote quadratic loss function:

$$Q(\mathbf{w}) = (Y - X\mathbf{w})^T(Y - X\mathbf{w}) = \|Y - X\mathbf{w}\|_2^2 \,,$$

where $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n], \quad \mathbf{x}_i \in \mathbb{R}^p \;\; Y = [y_1, \ldots, y_n], \quad y_i \in \mathbb{R}$.

To find optimal solution let's equal to zero the derivative of the equation above:

$$\nabla_{\mathbf{w}} Q(\mathbf{w}) = \nabla_{\mathbf{w}} [Y^T Y - Y^T X\mathbf{w} - \mathbf{w}^T X^T Y + \mathbf{w}^T X^T X\mathbf{w}] =$$

$$= 0 - X^T Y - X^T Y + (X^T X + X^T X)\mathbf{w} = 0$$

$$\hat{\mathbf{w}} = \boxed{(X^T X)}^{-1} X^T Y$$

what if this matrix is *singular*?

Analytical solution

$$\hat{\mathbf{w}} = \boxed{(X^T X)}^{-1} X^T Y$$

what if this matrix is *singular?*

Unstable solution

In case of multicollinear features the matrix $X^T X$ is almost singular .

It leads to unstable solution:

```
w_true
```

```
array([ 2.68647887, -0.52184084, -1.12776533])
```

```
w_star = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
w_star
```

```
array([   2.68027723, -186.0552577 ,  184.41701118])
```

Unstable solution

In case of multicollinear features the matrix $X^T X$ is almost singular .

It leads to unstable solution:

```
w_true
```

```
array([ 2.68647887, -0.52184084, -1.12776533])
```

```
w_star = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
w_star
```

```
array([   2.68027723, -186.0552577 ,  184.41701118])
```

corresponding features are almost collinear

Unstable solution

In case of multicollinear features the matrix $X^T X$ is almost singular .

It leads to unstable solution:

```
w_true
```

```
array([ 2.68647887, -0.52184084, -1.12776533])
```

```
w_star = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
w_star
```

```
array([   2.68027723, -186.0552577 ,  184.41701118])
```

the coefficients are huge and sum up to almost 0

Regularization

To make the matrix nonsingular, we can add a diagonal matrix:

$$\hat{\mathbf{w}} = (X^T X + \lambda I)^{-1} X^T Y \quad ,$$

To make the matrix nonsingular, we can add a diagonal matrix:

$$\hat{\mathbf{w}} = (X^T X + \lambda I)^{-1} X^T Y \quad ,$$

where $I = \mathrm{diag}[1_1, \ldots, 1_p]$.

Regularization

To make the matrix nonsingular, we can add a diagonal matrix:

$$\hat{\mathbf{w}} = (X^T X + \lambda I)^{-1} X^T Y \quad ,$$

where $I = \mathrm{diag}[1_1, \ldots, 1_p]$.

Actually, it's a solution for the following loss function:

$$Q(\mathbf{w}) = \|Y - X\mathbf{w}\|_2^2 + \lambda^2 \|\mathbf{w}\|_2^2$$

exercise: derive it by yourself

Gauss-Markov theorem

Gauss-Markov theorem

Suppose target values are expressed in following form:

$$Y = X\mathbf{w} + \boldsymbol{\varepsilon}$$, where $$\boldsymbol{\varepsilon} = [\varepsilon_1, \ldots, \varepsilon_N]$$

are random variables

Gauss–Markov assumptions:

- $\mathbb{E}(\varepsilon_i) = 0 \quad \forall i$
- $\mathrm{Var}(\varepsilon_i) = \sigma^2 < \inf \quad \forall i$
- $\mathrm{Cov}(\varepsilon_i, \varepsilon_j) = 0 \quad \forall i \neq j$

Gauss-Markov theorem

- $\mathbb{E}(\varepsilon_i) = 0 \quad \forall i$
- $\mathrm{Var}(\varepsilon_i) = \sigma^2 < \inf \quad \forall i$
- $\mathrm{Cov}(\varepsilon_i, \varepsilon_j) = 0 \quad \forall i \neq j$

$$\boxed{\hat{\mathbf{w}} = (X^T X)^{-1} X^T Y}$$

delivers **B**est **L**inear **U**nbiased **E**stimator

Loss functions in regression

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$
$$MAE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|_1 = \frac{1}{N}\sum_i |y_i - \hat{y}_i|$$

Different norms

Once more: loss functions:

$$\boxed{MSE = \frac{1}{n}\|\mathbf{x}^T\mathbf{w} - \mathbf{y}\|_2^2}$$

only works for Gauss-Markov theorem

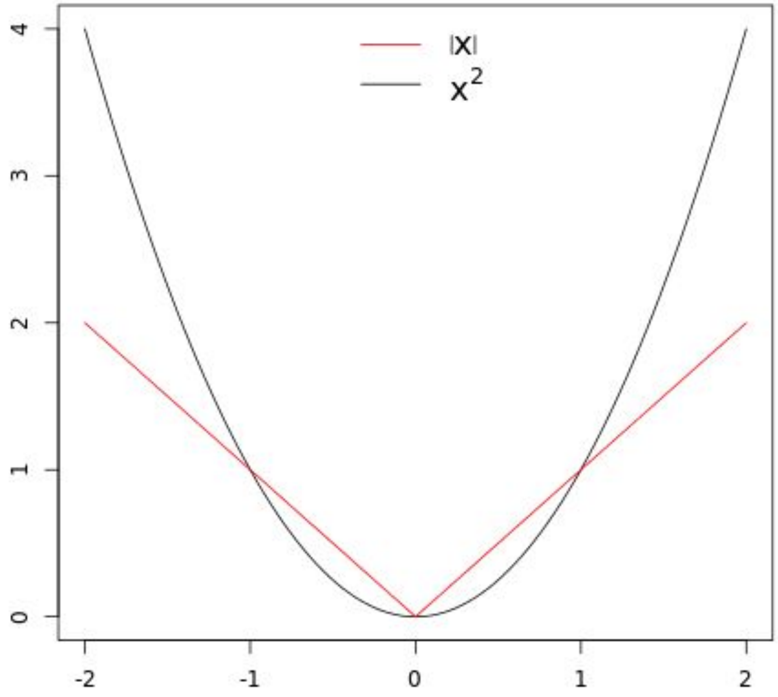
$$MAE = \frac{1}{n}\|\mathbf{x}^T\mathbf{w} - \mathbf{y}\|_1$$

Regularization terms:

- $L_2 : \|\mathbf{w}\|_2^2$
- $L_1 : \|\mathbf{w}\|_1$

What's the difference?

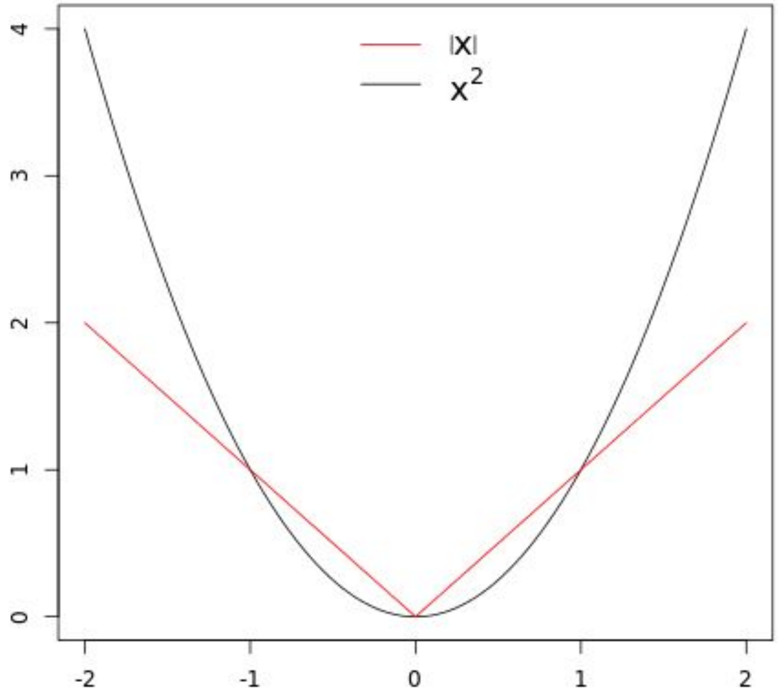
- MSE (L_2)
  - delivers BLUE according to Gauss-Markov theorem
  - differentiable
  - sensitive to noise
- MAE (L_1)
  - non-differentiable
    - not a problem
  - much more prone to noise



What's the difference?

- L_2 regularization
  - constraints weights
  - delivers more stable solution
  - differentiable

- L_1 regularization
  - non-differentiable
    - not a problem
  - selects features



Loss functions in regression

Other functions to measure the quality in regression:

- R2 score
- MAPE
- SMAPE
- ...

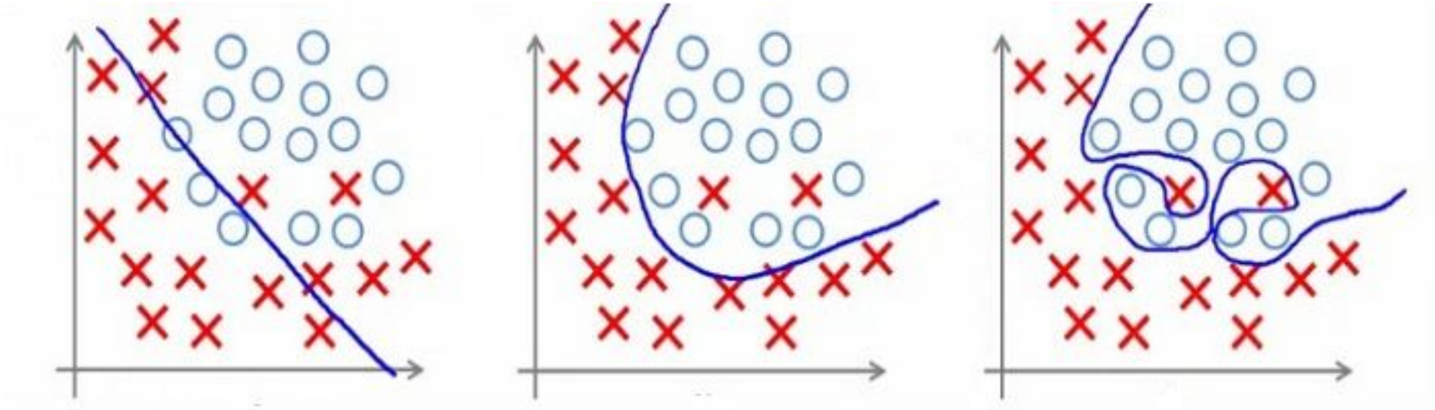
Model validation and evaluation

Supervised learning problem statement

Let's denote:

- Training set $\mathcal{L} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, where
  - $(\mathbf{x} \in \mathbb{R}^p,\ y \in \mathbb{R}\ )$ for regression
  - $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{+1, -1\}$ for binary classification
- Model $f(\mathbf{x})$ predicts some value for every object
- Loss function $Q(\mathbf{x}, y, f)$ that should be minimized

Overfitting vs. underfitting



Under-fitting

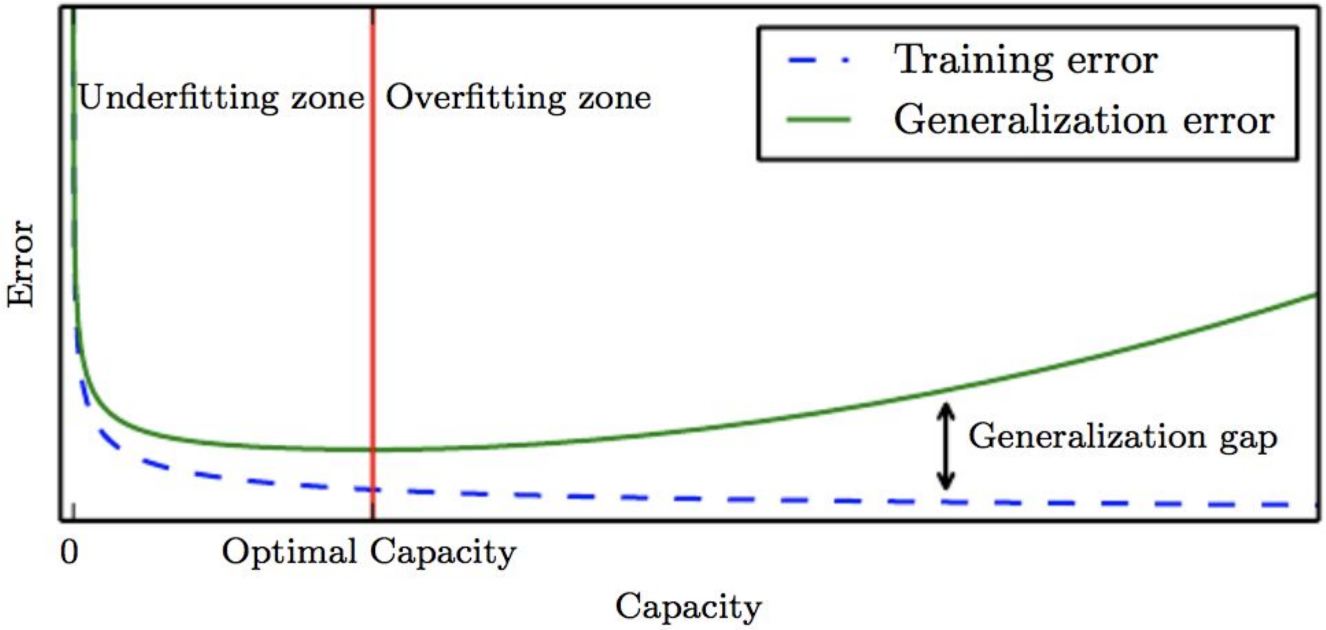
(too simple to explain the variance)

Appropriate-fitting

Over-fitting

(forcefitting -- too good to be true)

Overfitting vs. underfitting



Overfitting vs. underfitting

- We can control overfitting / underfitting by altering model’s capacity (ability to fit a wide variety of functions):
- select appropriate hypothesis space
- learning algorithm’s effective capacity may be less than the representational capacity of the model family

Evaluating the quality

Image credit: Joseph Nelson [@josephofiowa]

Evaluating the quality

Is it good enough?

Image credit: Joseph Nelson @josephofiowa

Evaluating the quality

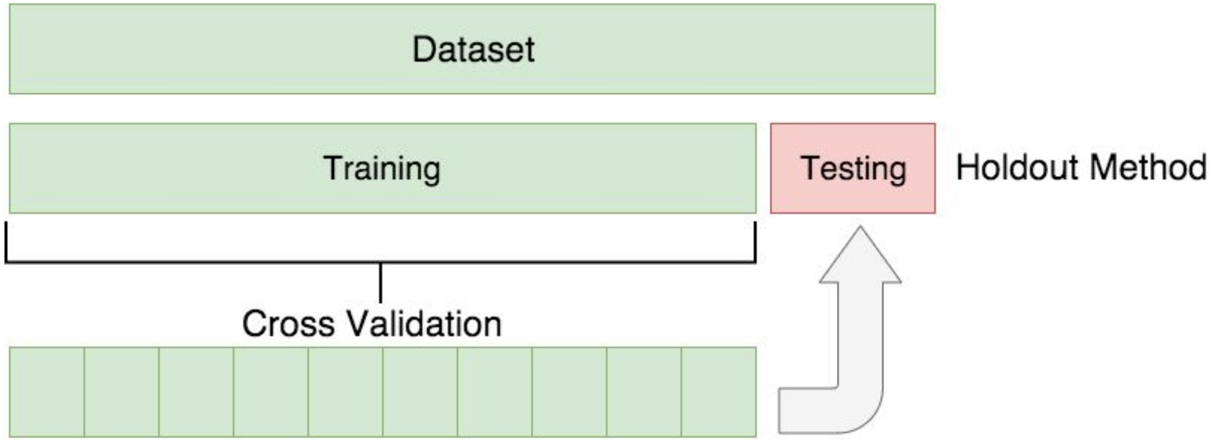


Image credit: Joseph Nelson @josephofiowa

Evaluating the quality

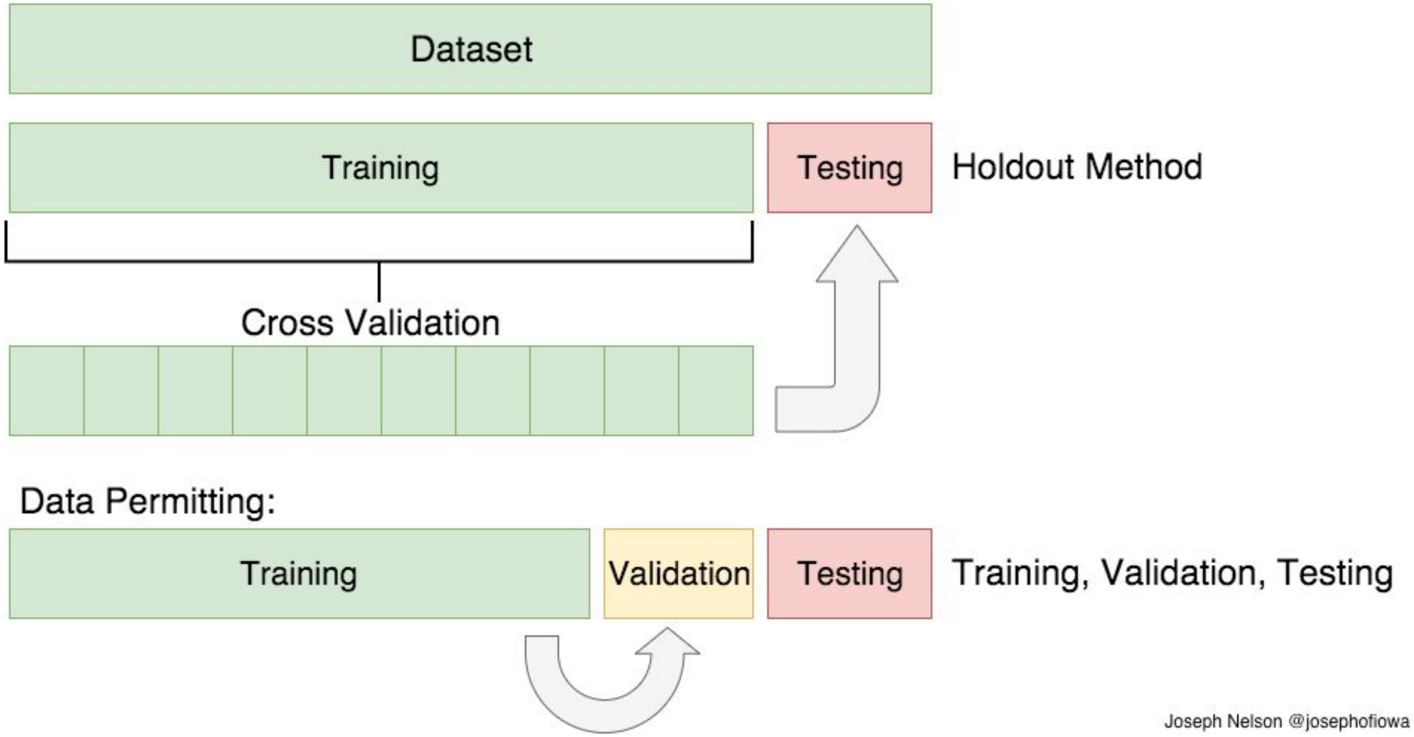
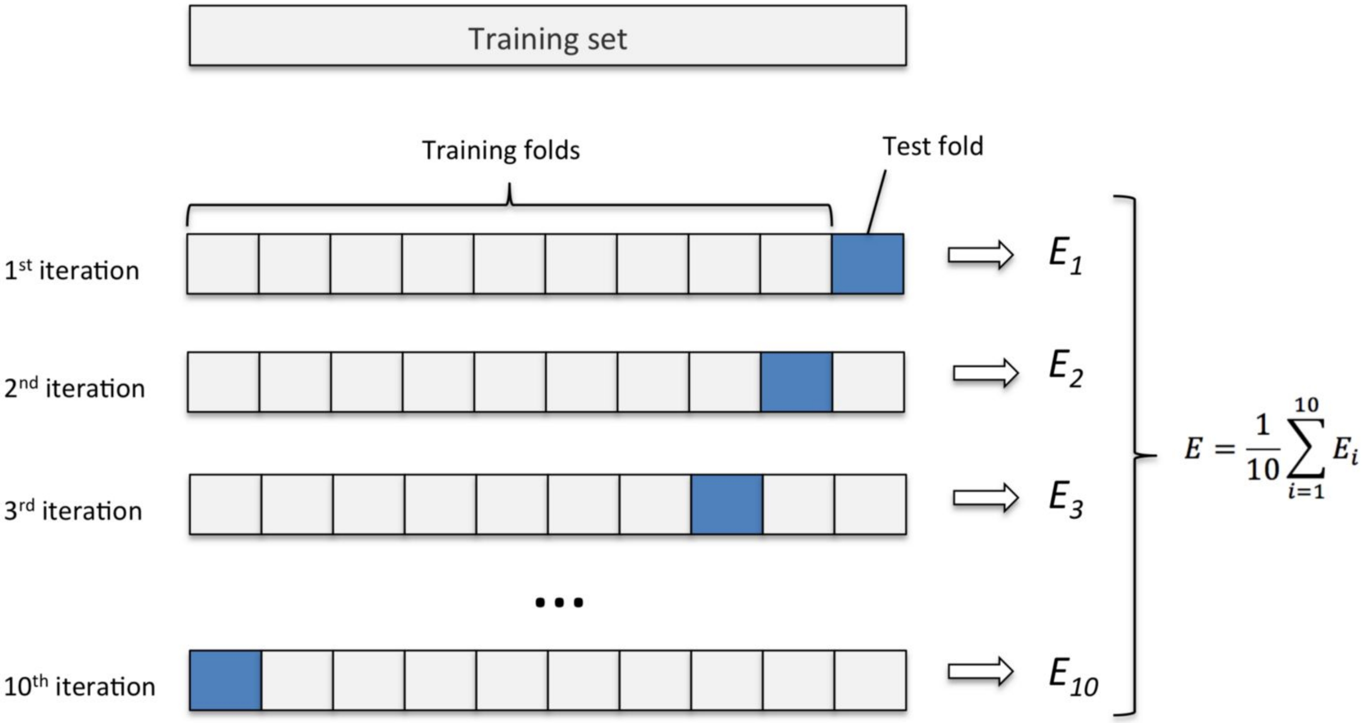


Image credit: Joseph Nelson [@josephofiowa]

Cross-validation



Backup