Faculty of Science

L3 – Non-Linear Regression

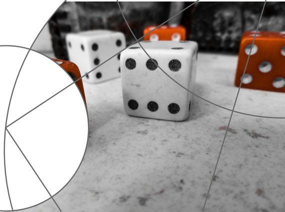
Modelling and Analysis of Data

Fabian Gieseke

Machine Learning Section
Department of Computer Science
University of Copenhagen

Universitetsparken 1, 1-1-N110
fabian.gieseke@di.ku.dk

26 November 2019



Outline

Outline

Multivariate Linear Regression

- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$.
- Let's "augment" all data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times (D+1)}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,D} \\ \vdots & & & & \\ 1 & x_{N,1} & x_{N,2} & \ldots & x_{N,D} \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- As before, we can write the overall loss in the following form:

Overall Loss

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \left(f(\mathbf{x}_n; \mathbf{w}) - t_n\right)^2 = \frac{1}{N} (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})$$

How can we obtain the optimal $(d+1)$-dimensional coefficient vector $\boldsymbol{w}$?

Summary: Multivariate Linear Regression

- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$.
- Goal: Find $(D+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_D]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})$, i.e., which is a solution for

$$\begin{aligned} \nabla \mathcal{L}(\mathbf{w}) &= \mathbf{0} \\ \Leftrightarrow \quad \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned} \tag{1}$$

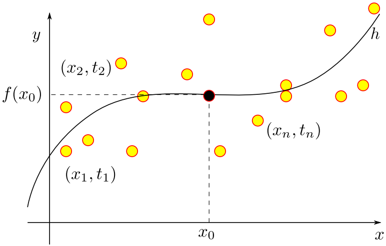
Computation in Practice

1. Definition of data matrix $\mathbf{X} \in \mathbb{R}^{N\times(D+1)}$
   (make use of Numpy arrays and functions!)
2. There are different ways to compute an optimal weight vector $\hat{\mathbf{w}}$:
   1. Compute $\hat{\mathbf{w}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$ (e.g., via `numpy.linalg.inv`)
   2. Directly solve system of equations (1) (e.g., via `numpy.linalg.solve`)
   3. ...
3. For new point $\mathbf{x}_{new} \in \mathbb{R}^D$: Compute $t_{new} = [1, \mathbf{x}_{new}^T]\hat{\mathbf{w}}$

New to Python/Numpy? $\rightarrow$
http://cs231n.github.io/python-numpy-tutorial

Outline

“Non-Linear” Models?



Quadratic Models

- Let's focus again on $D = 1$, i.e., on input data of the form $x_n \in \mathbb{R}$.

Quadratic Models

- Let's focus again on $D = 1$, i.e., on input data of the form $x_n \in \mathbb{R}$.
- Now, let's "augment" all data points $x_1, x_2, \ldots, x_N$, now with an additional column containing x_n^2. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times 3}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_N & x_N^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

Quadratic Models

- Let's focus again on $D = 1$, i.e., on input data of the form $x_n \in \mathbb{R}$.
- Now, let's "augment" all data points $x_1, x_2, \ldots, x_N$, now with an additional column containing x_n^2. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times 3}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_N & x_N^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Note: Basically as before, just a "new" input feature/variable.
(i.e., we have generated a new column based on an existing column)

Quadratic Models

- Let's focus again on $D = 1$, i.e., on input data of the form $x_n \in \mathbb{R}$.
- Now, let's "augment" all data points $x_1, x_2, \ldots, x_N$, now with an additional column containing x_n^2. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times 3}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_N & x_N^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Note: Basically as before, just a "new" input feature/variable.
  (i.e., we have generated a new column based on an existing column)
- As before: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}$ with $\boldsymbol{x} = [1, x, x^2]^T$ and $\boldsymbol{w} = [w_0, w_1, w_2]^T$

Quadratic Models

- Let's focus again on $D = 1$, i.e., on input data of the form $x_n \in \mathbb{R}$.
- Now, let's "augment" all data points $x_1, x_2, \ldots, x_N$, now with an additional column containing x_n^2. This yields an augmented data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times 3}$ and an associated target vector $\boldsymbol{t} \in \mathbb{R}^N$:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & & \\ 1 & x_N & x_N^2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Note: Basically as before, just a "new" input feature/variable. (i.e., we have generated a new column based on an existing column)
- As before: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}$ with $\boldsymbol{x} = [1, x, x^2]^T$ and $\boldsymbol{w} = [w_0, w_1, w_2]^T$
- Our model is still linear in the parameters, but the actual function that is fitted is now quadratic:

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2$$

Polynomial Models

- We can continue adding columns of this form ...

$$\boldsymbol{X} = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \dots & x_1^K \\ x_2^0 & x_2^1 & x_2^2 & \dots & x_2^K \\ \vdots & & & & \\ x_N^0 & x_N^1 & x_N^2 & \dots & x_N^K \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Our model function can then be written as $f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{k=0}^{K} w_k x^k$

Coding Time!

jupyter Non_Linear_Regression Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3

Run Code

move selected cells down

In [ ]:
```python
import numpy
import matplotlib.pyplot as plt
import linreg
```

In [ ]:
```python
# number of data points
n_points = 15

# set a seed here to initialize the random number generator
# (such that we get the same dataset each time this cell is executed)
numpy.random.seed(1)

# let's generate some "non-linear" data; note
# that the sorting step is done for visualization
# purposes only (to plot the models as connected lines)
X = numpy.random.uniform(-10,10, n_points)
t = X**2 + numpy.random.random(n_points) * 25

# reshape both arrays to make sure that we deal with
# N-dimensional Numpy arrays
t = t.reshape((len(t), 1))
X = X.reshape((len(X),1))
print("Shape of our data matrix: %s" % str(X.shape))
print("Shape of our target vector: %s" % str(t.shape))
```

In [ ]:
```python
# instantiate the regression model
model = linreg.LinearRegression()

# fit the model
model.fit(X,t)

# get predictions for the data points
preds = model.predict(X)
```

Arbitrary "Basis Functions"

- We can basically resort to arbitrary functions ...

$$\boldsymbol{X} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & h_3(x_1) & \ldots & h_K(x_1) \\ h_1(x_2) & h_2(x_2) & h_3(x_2) & \ldots & h_K(x_2) \\ \vdots & & & & \\ h_1(x_N) & h_2(x_N) & h_3(x_N) & \ldots & h_K(x_N) \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

- Our model function can then be written as $f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{k=0}^{K} w_k h_k(x)$

Arbitrary "Basis Functions"

- We can basically resort to arbitrary functions ...

$$\boldsymbol{X} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & h_3(x_1) & \ldots & h_K(x_1) \\ h_1(x_2) & h_2(x_2) & h_3(x_2) & \ldots & h_K(x_2) \\ \vdots & & & & \\ h_1(x_N) & h_2(x_N) & h_3(x_N) & \ldots & h_K(x_N) \end{bmatrix} \quad \text{and} \quad \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

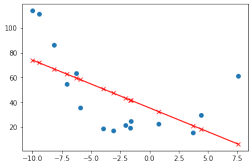
- Our model function can then be written as $f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{k=0}^{K} w_k h_k(x)$

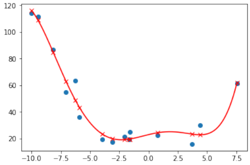
General Case

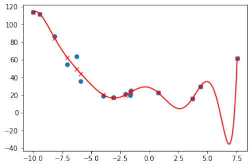
Also, given more input variables ($D > 1$), we can simply

- transform each input variable/column ...
- combine different input variables (e.g., difference between columns) ...
- combine and transform input variables ...
- ...

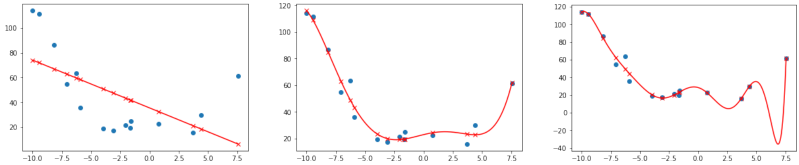
Which model is the best?





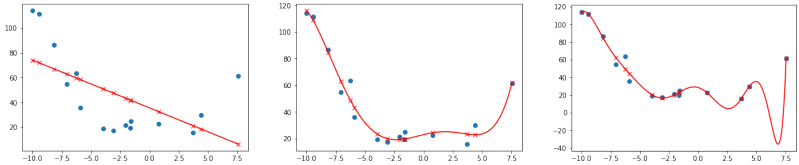


Which model is the best?



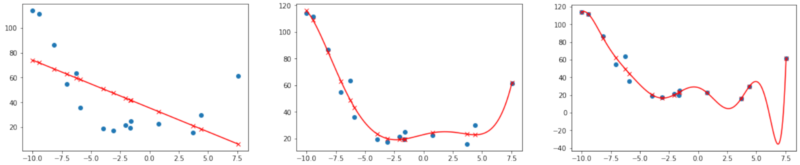
- We are given a so-called training set $T = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \subset \mathbb{R}^D \times \mathbb{R}$.

Which model is the best?



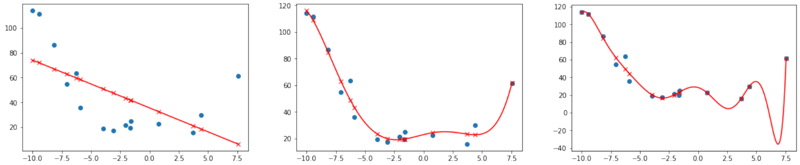
- We are given a so-called training set $T = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \subset \mathbb{R}^D \times \mathbb{R}$.
- Given the additional flexibility, we now have to choose a “good” model . . .
  1. Which non-linear functions should we choose?
  2. How many additional columns should be generated?
  3. . . .

Which model is the best?



- We are given a so-called training set $T = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \subset \mathbb{R}^D \times \mathbb{R}$.
- Given the additional flexibility, we now have to choose a “good” model . . .
  1. Which non-linear functions should we choose?
  2. How many additional columns should be generated?
  3. . . .
- We would like to choose a model that performs well on new, unseen data!

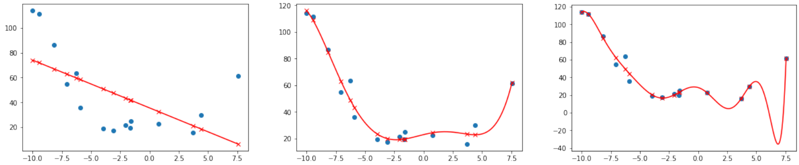
Which model is the best?



- We are given a so-called training set $T = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \subset \mathbb{R}^D \times \mathbb{R}$.
- Given the additional flexibility, we now have to choose a "good" model ...
  1. Which non-linear functions should we choose?
  2. How many additional columns should be generated?
  3. ...
- We would like to choose a model that performs well on new, unseen data!

Question: How can we select such a model?

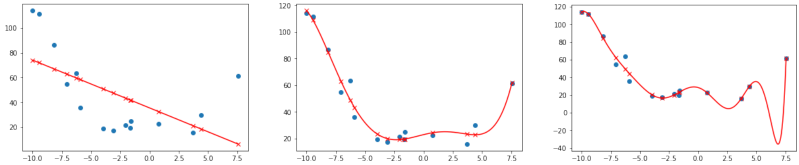
Evaluation of “Model Quality”



- The quality can be evaluated using, e.g., the mean squared error (MSE):

$$\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n; \mathbf{w})\right)^2$$

Evaluation of “Model Quality”



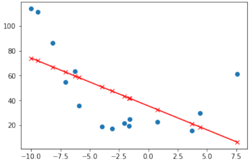
- The quality can be evaluated using, e.g., the mean squared error (MSE):

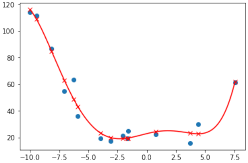
$$\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2$$

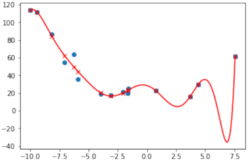
- A variant is the so-called root mean squared error (RMSE):

$$\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2}$$

Evaluation of "Model Quality"







- The quality can be evaluated using, e.g., the mean squared error (MSE):

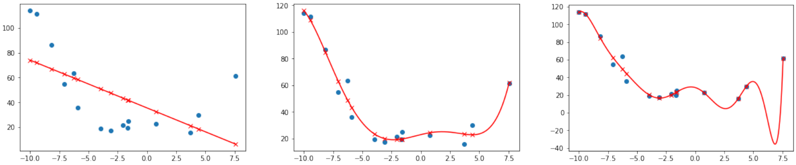
$$\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2$$

- A variant is the so-called root mean squared error (RMSE):

$$\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2}$$

- How should we evaluate the error?

Evaluation of “Model Quality”



- The quality can be evaluated using, e.g., the mean squared error (MSE):

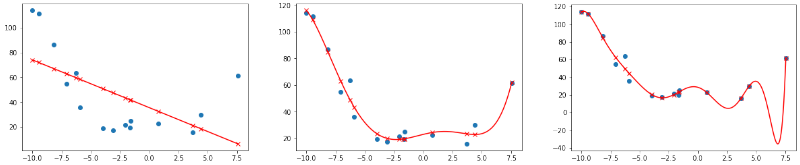
$$\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2$$

- A variant is the so-called root mean squared error (RMSE):

$$\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2}$$

- How should we evaluate the error? On a separate dataset!

Evaluation of "Model Quality"



- The quality can be evaluated using, e.g., the mean squared error (MSE):

$$\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2$$

- A variant is the so-called root mean squared error (RMSE):

$$\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(t_n - f(\mathbf{x}_n;\mathbf{w})\right)^2}$$

- How should we evaluate the error? On a separate dataset! Why?

Coding Time!

jupyter Non_Linear_Regression_Training_Validation (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Code

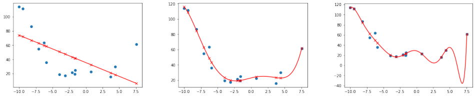
In [1]:
```
import numpy
import matplotlib.pyplot as plt
import linreg
```

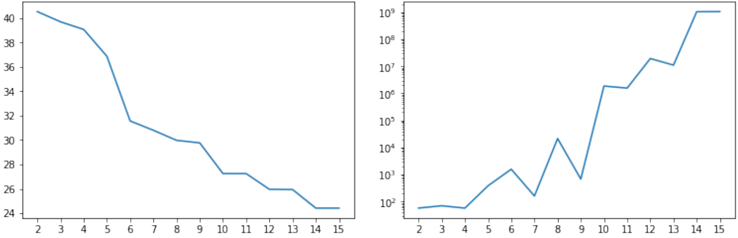
In [2]:
```
# set a seed here to initialize the random number generator
# (such that we get the same dataset each time this cell is executed)
numpy.random.seed(2)

# note: using large values here might lead to numerical inaccuracies
order_range = range(2,16)

# let's generate some "non-linear" data; not
# that the sorting step is done for visualization
# purposes only (to plot the models as connected lines)
X = numpy.random.uniform(-10,10,80)
t = X**2 + numpy.random.random(80) * 25

# reshape both arrays to make sure that we deal with
# N-dimensional Numpy arrays
X = X.reshape((len(X),1))
t = t.reshape((len(t), 1))
print("Shape of our data matrix: %s" % str(X.shape))
print("Shape of our target vector: %s" % str(t.shape))
```

```
Shape of our data matrix: (80, 1)
Shape of our target vector: (80, 1)
```

In [3]:
```
def augment(X, max_order):
    """ Augments a given data
    matrix by adding additional
    columns.

    NOTE: In case max_order is very large,
    numerical inaccuracies might occur
    """
```

Overfitting

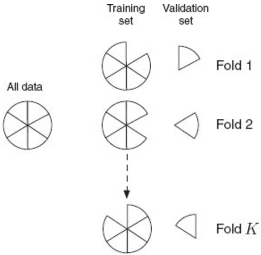




Training error (left) and validation error (right) when fitting polynomials of increasing order (x-axis).

Overfitting: Good training error, bad validation error.

Training and Validation Sets



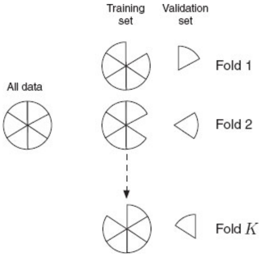
Cross-Validation

K-fold cross-validation splits the data into K (almost) equally-sized parts. We consider K "rounds":

1. Use $K-1$ parts for training the model. For instance, the optimal weight vector $\mathbf{w}$ is computed for linear regression.
2. Use the remaining part for validating the model by computing the induced loss on this part.

This yields K validation errors. Typically, the average of these values is considered.

Training and Validation Sets



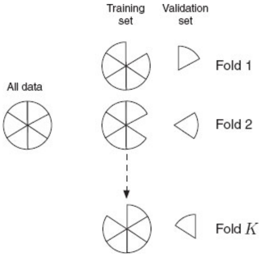
Cross-Validation

Typical values for K are $K = 2$, $K = 5$, $K = 10$, or $K = N$. The last case ($K = N$) is called Leave-One-Out Cross Validation (LOOCV). The average validation error for LOOCV is given by

$$\mathcal{L}^{CV} = \frac{1}{N} \sum_{n=1}^{N} (t_n - f(\mathbf{x}_n; \mathbf{w}_{-n}))^2$$

where $\mathbf{w}_{-n}$ is weight vector computed without the n-th training example.

Training and Validation Sets



Cross-Validation

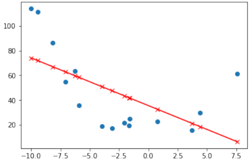
Typical values for K are $K = 2$, $K = 5$, $K = 10$, or $K = N$. The last case ($K = N$) is called Leave-One-Out Cross Validation (LOOCV). The average validation error for LOOCV is given by

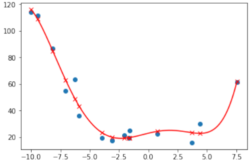
$$\mathcal{L}^{CV} = \frac{1}{N} \sum_{n=1}^{N} (t_n - f(\mathbf{x}_n; \mathbf{w}_{-n}))^2$$

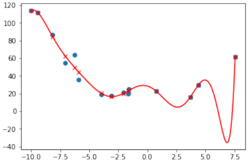
where $\mathbf{w}_{-n}$ is weight vector computed without the n-th training example.

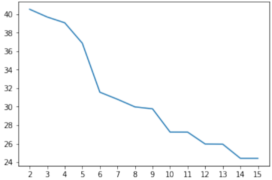
Question: Advantages of $K = 2$, $K = 10$, or $K = N$?

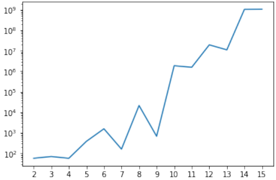
Which model is the best?





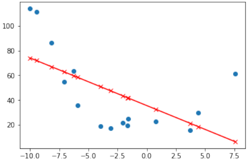


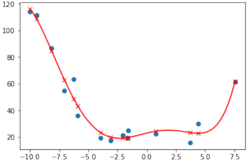


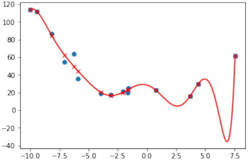


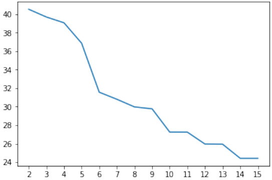
Question: How can we select such a model?

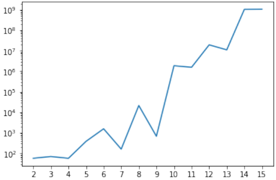
Which model is the best?











Question: How can we select such a model?

Select the model with the best validation error!

Real-World Performance

Final model performance?

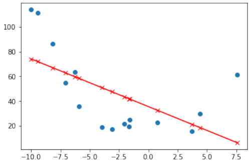
Real-World Performance

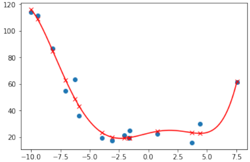
Final model performance?

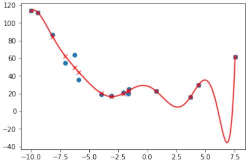
Make use of a third dataset, the so-called test set, for the final evaluation!

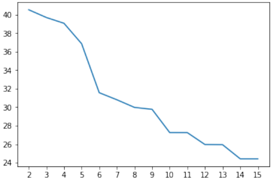
Outline

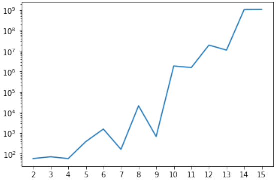
Which Model is the Best?





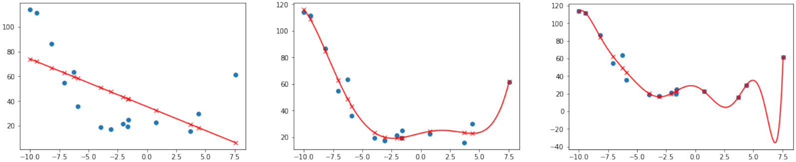






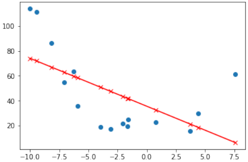
One option: Stop increasing model complexity as soon as model becomes too complex, i.e., when it starts over-fitting (training error goes down, but validation error goes up!).

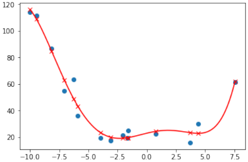
Regularisation

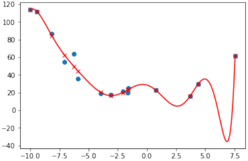


- The simple model $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T \mathbf{w}$ with $\mathbf{w} = [0, \ldots, 0]^T$ always predicts 0.

Regularisation





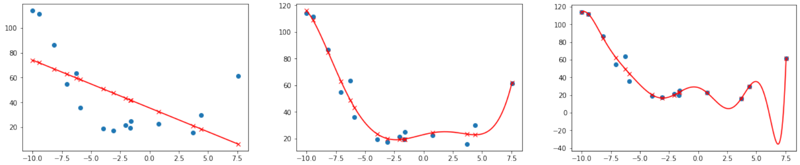


- The simple model $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T \mathbf{w}$ with $\mathbf{w} = [0, \ldots, 0]^T$ always predicts 0.
- Consider the following 5-th order polynomial:

$$f(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5$$

Let's start with $\mathbf{w} = \mathbf{0}$. Now, by allowing some of the w_i to be non-zero, we can make the model more and more complex/flexible!

Regularisation



- The simple model $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T \mathbf{w}$ with $\mathbf{w} = [0, \ldots, 0]^T$ always predicts 0.
- Consider the following 5-th order polynomial:

$$f(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5$$

  Let's start with $\mathbf{w} = \mathbf{0}$. Now, by allowing some of the w_i to be non-zero, we can make the model more and more complex/flexible!
- Remember, we don't want too complex models! To avoid this, we can "penalize" complex models by adding the term $\sum_i w_i^2$ to the objective:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}) + \lambda \mathbf{w}^T \mathbf{w},$$

  where $\lambda > 0$ is a model parameter controlling the trade-off between penalising (a) not fitting the data well and (b) overly complex models.

Gradient

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t} + \lambda\mathbf{w}^T\mathbf{w}$$

Gradient

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t} + \lambda\mathbf{w}^T\mathbf{w}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{x} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
2. $f(\mathbf{w}) = \boldsymbol{x}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
3. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{w}$
4. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{C}\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{C}\boldsymbol{w}$ (if $\boldsymbol{C}$ is symmetric)

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w}$ and therefore:

$$\frac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w} \quad = \mathbf{0}$$

Gradient

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t} + \lambda\mathbf{w}^T\mathbf{w}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{x} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
2. $f(\mathbf{w}) = \boldsymbol{x}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
3. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{w}$
4. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{C}\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{C}\boldsymbol{w}$ (if $\boldsymbol{C}$ is symmetric)

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w}$ and therefore:

$$\begin{aligned} & \tfrac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \tfrac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w} &= \mathbf{0} \\ \Leftrightarrow \quad & \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} + N\lambda\mathbf{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned}$$

Gradient

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t} + \lambda\mathbf{w}^T\mathbf{w}$$

Toolbox (Table 1.4 in Rogers & Girolami)

1. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{x} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
2. $f(\mathbf{w}) = \boldsymbol{x}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
3. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{w}$
4. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{C}\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{C}\boldsymbol{w}$ (if $\boldsymbol{C}$ is symmetric)

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w}$ and therefore:

$$\begin{aligned}
& \tfrac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \tfrac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w} && = \mathbf{0} \\
\Leftrightarrow \quad & \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} + N\lambda\mathbf{w} && = \boldsymbol{X}^T\boldsymbol{t} \\
\Leftrightarrow \quad & (\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})\boldsymbol{w} && = \boldsymbol{X}^T\boldsymbol{t}
\end{aligned}$$

Gradient

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t} + \lambda\mathbf{w}^T\mathbf{w}$$

Toolbox (Table 1.4 in Rogers & Girolami)

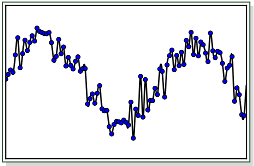
1. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{x} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
2. $f(\mathbf{w}) = \boldsymbol{x}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = \boldsymbol{x}$
3. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{w} \Rightarrow \nabla f(\boldsymbol{w}) = 2\boldsymbol{w}$
4. $f(\mathbf{w}) = \boldsymbol{w}^T\boldsymbol{C}\boldsymbol{w} \Rightarrow \nabla f(\mathbf{w}) = 2\boldsymbol{C}\boldsymbol{w}$ (if $\boldsymbol{C}$ is symmetric)

We have $\nabla\mathcal{L}(\mathbf{w}) = \frac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \frac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w}$ and therefore:

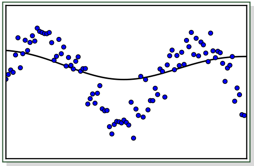
$$\begin{aligned} & \tfrac{2}{N}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \tfrac{2}{N}\boldsymbol{X}^T\boldsymbol{t} + 2\lambda\mathbf{w} && = \mathbf{0} \\ \Leftrightarrow \quad & \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} + N\lambda\mathbf{w} && = \boldsymbol{X}^T\boldsymbol{t} \\ \Leftrightarrow \quad & (\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})\boldsymbol{w} && = \boldsymbol{X}^T\boldsymbol{t} \end{aligned}$$

Small comment: The matrix $(\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})$ is always invertible.

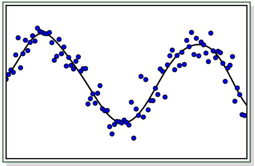
Regularised Linear Regression



$\lambda =$ small



$\lambda =$ large



$\lambda =$ middle

Minimizer for Regularised Linear Regression

$$\hat{\mathbf{w}} = \left(\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{t}$$

Coding Time!

jupyter Non_Linear_Regression_Regularisation (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted Python 3

Run Code

copy selected cells

In [ ]:
```
import numpy
import matplotlib.pyplot as plt
import linreg
```

In [ ]:
```
# number of data points
n_points = 10

# set a seed here to initialize the random number generator
# (such that we get the same dataset each time this cell is executed)
numpy.random.seed(1)

# let's generate some "non-linear" data; note
# that the sorting step is done for visualization
# purposes only (to plot the models as connected lines)
X = numpy.random.uniform(-10,10, n_points)
t = X**2 + numpy.random.random(n_points) * 25

# reshape both arrays to make sure that we deal with
# N-dimensional Numpy arrays
t = t.reshape((len(t), 1))
X = X.reshape((len(X),1))
print("Shape of our data matrix: %s" % str(X.shape))
print("Shape of our target vector: %s" % str(t.shape))
```

In [ ]:
```
# maximum degree
max_degree = 7

def augment(X, max_order):
    """ Augments a given data
    matrix by adding additional
    columns.

    NOTE: In case max_order is very large,
    numerical inaccuracies might occur
    """

    X_augmented = X
```

Multivariate Regularised Linear Regression

- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$ and parameter $\lambda > 0$.
- Goal: Find $(D+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_D]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}) + \lambda \mathbf{w}^T\mathbf{w}$, i.e., which is a solution to

$$\begin{aligned} \nabla \mathcal{L}(\mathbf{w}) &= \mathbf{0} \\ \Leftrightarrow \quad (\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned} \tag{2}$$

New to Python/Numpy? $\rightarrow$
http://cs231n.github.io/python-numpy-tutorial

Multivariate Regularised Linear Regression

- Given: Pairs of the form $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^D \times \mathbb{R}$ and parameter $\lambda > 0$.
- Goal: Find $(D+1)$-dimensional weight vector $\hat{\mathbf{w}} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_D]^T$ that minimizes $\mathcal{L}(\mathbf{w}) = \frac{1}{N}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}) + \lambda \mathbf{w}^T\mathbf{w}$, i.e., which is a solution to

$$\begin{aligned} \nabla \mathcal{L}(\mathbf{w}) &= \mathbf{0} \\ \Leftrightarrow \quad (\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})\boldsymbol{w} &= \boldsymbol{X}^T\boldsymbol{t} \end{aligned} \tag{2}$$

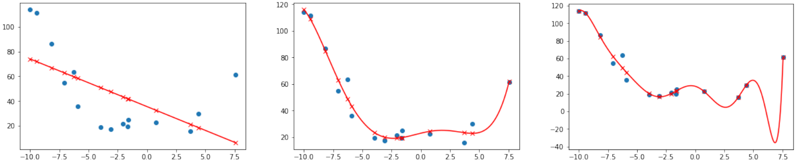
Computation in Practice

1. Definition of data matrix $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$
   (make use of Numpy arrays and functions!)
2. There are different ways to compute an optimal weight vector $\hat{\mathbf{w}}$:
   1. Compute $\hat{\mathbf{w}} = (\boldsymbol{X}^T\boldsymbol{X} + N\lambda\mathbf{I})^{-1}\boldsymbol{X}^T\boldsymbol{t}$ (e.g., via `numpy.linalg.inv`)
   2. Directly solve system of equations (2) (e.g., via `numpy.linalg.solve`)
   3. ...
3. For new point $\mathbf{x}_{new} \in \mathbb{R}^D$: Compute $t_{new} = [1, \mathbf{x}_{new}^T]\hat{\mathbf{w}}$

New to Python/Numpy? →
http://cs231n.github.io/python-numpy-tutorial

Outline

Summary & Outlook



Today

- Recap: Multivariate linear regression
- Non-linear models by augmenting data matrix
- Cross-validation: How to select a good model . . .
- Regularisation: How to avoid overfitting . . .

Outlook

- Statistics (Thursday, Bulat)