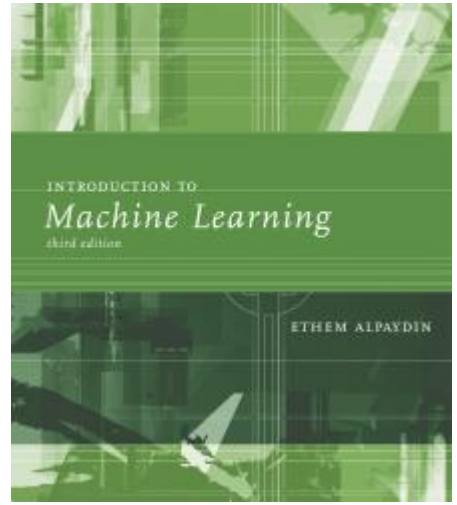



# Polynomial Regression & Regularization

Lecture notes by Ethem Alpaydın
Introduction to Machine Learning (Boğaziçi Üniversitesi)

Lecture notes by Kevyn Collins-Thompson
Applied Machine Learning (Coursera)

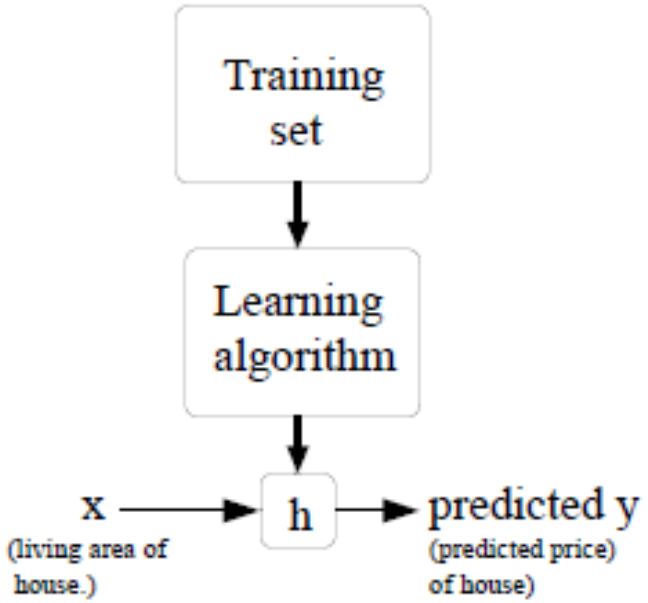
Lecture notes by Andrew NG
Machine Learning by Stanford University (Coursera)

# Revisiting Supervised Learning

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function $\mathcal{H}: \mathcal{X} \rightarrow \mathcal{Y}$ so that $\mathcal{H}(x)$ is a "good" predictor for the corresponding value of $y$.

($\mathcal{X}$ denote the space of input values, and $\mathcal{Y}$ the space of output values)

For historical reasons, this function $\mathcal{H}$ is called a hypothesis.



# Model Selection & Generalization

- Learning is an ill-posed problem; data is not sufficient to find a unique solution
- We should make some extra assumptions to have a unique solution with the data we have.
- The set of assumptions we make to have learning possible is called the *inductive bias* of the learning algorithm.

# Model Selection & Generalization

- The need for inductive bias, assumptions about $\mathcal{H}$
- Thus learning is not possible without inductive bias, and now the question is how to choose the right bias.
- This is called *model selection*, which is choosing between possible $\mathcal{H}$

# Model Selection & Generalization

- How well a model trained on the training set predicts the right output for new generalization instances is called *generalization*.
- Overfitting: $\mathcal{H}$ is more complex than the function
- Underfitting: $\mathcal{H}$ is less complex than the function

# Polynomial Regression – one variable

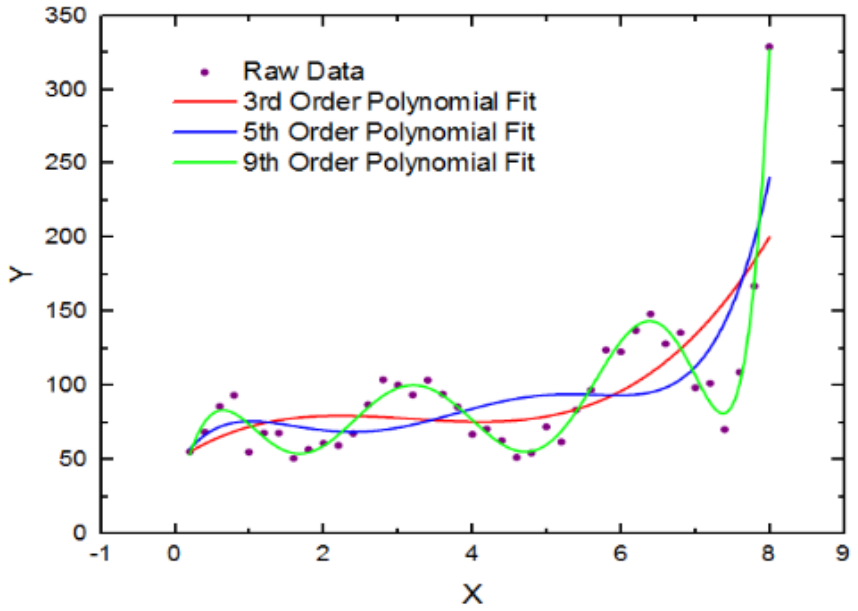
$$g\left(x^t \mid w_k,\ldots,w_2,w_1,w_0\right)=w_k\left(x^t\right)^k+\cdots+w_2\left(x^t\right)^2+w_1x^t+w_0$$

$$\mathbf{D}=\begin{bmatrix}1 & x^1 & \left(x^1\right)^2 & \cdots & \left(x^1\right)^k\\ 1 & x^2 & \left(x^2\right)^2 & \cdots & \left(x^2\right)^k\\ \vdots & & & & \\ 1 & x^N & \left(x^N\right)^2 & \cdots & \left(x^N\right)^2\end{bmatrix}\quad \mathbf{r}=\begin{bmatrix}r^1\\ r^2\\ \vdots\\ r^N\end{bmatrix}$$

$$\mathbf{w}=\left(\mathbf{D}^T\mathbf{D}\right)^{-1}\mathbf{D}^T\mathbf{r}$$

# Polynomial Regression – one variable

$$g(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 \quad \rightarrow 3rd\ order$$
$$g(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 \quad \rightarrow 5th\ order$$



# Bias/Variance Dilemma

- As we increase complexity,

  bias decreases (a better fit to data) and

  variance increases (fit varies more with data)

# Bias/Variance Dilemma

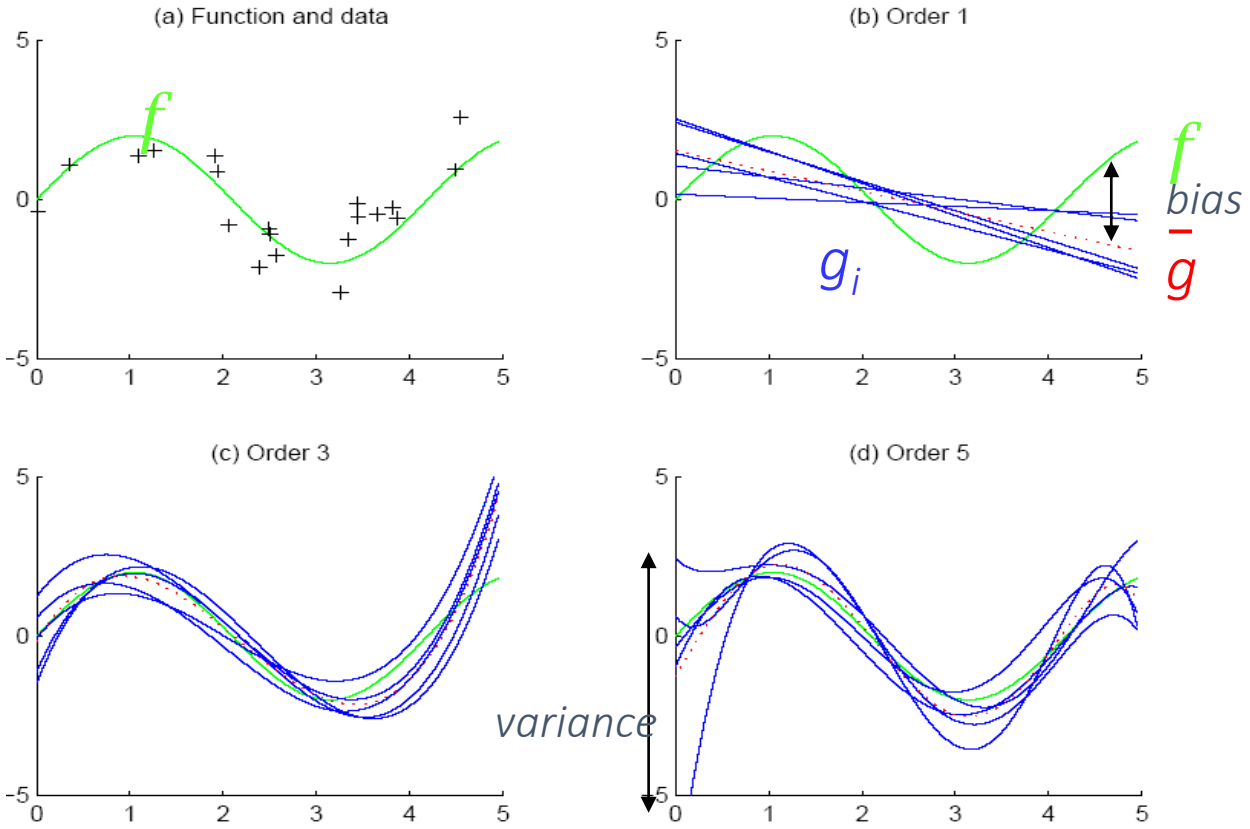
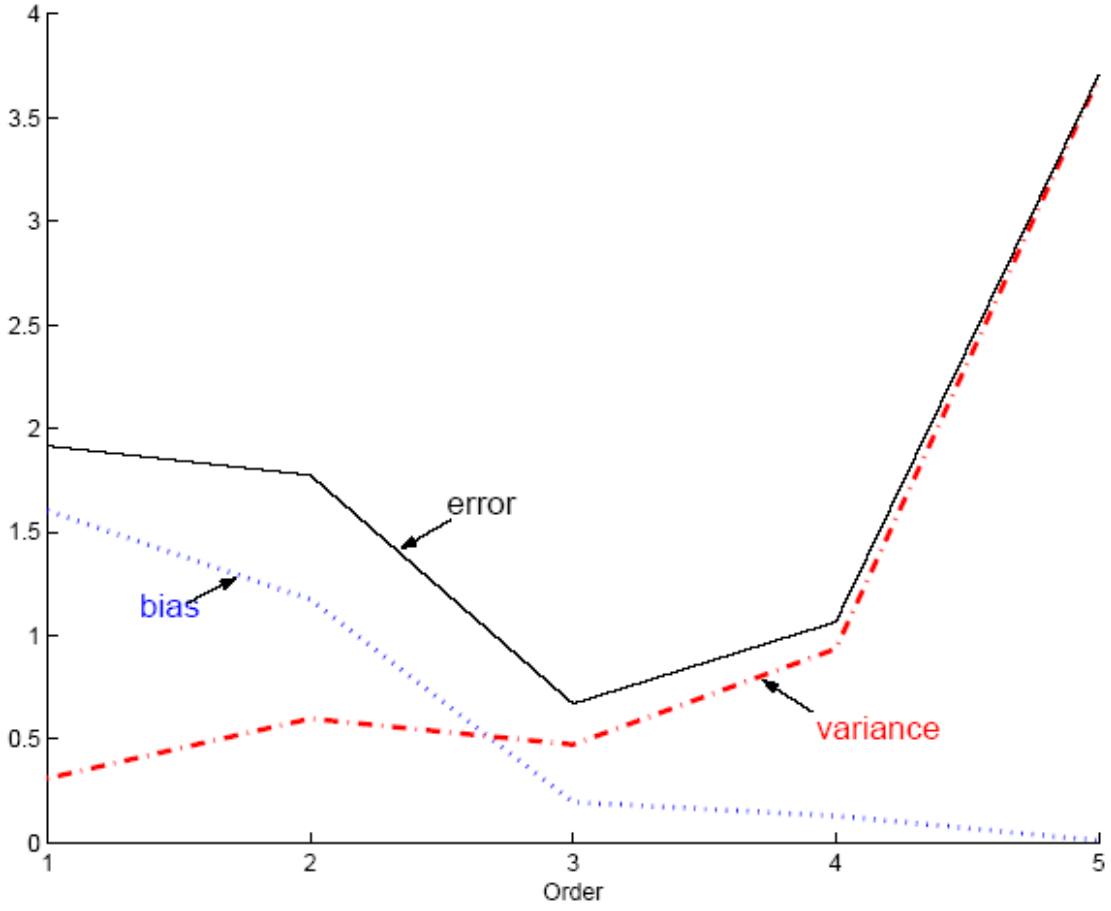



**Figure 4.5** (a) Function, $f(x) = 2\sin(1.5x)$, and one noisy ($\mathcal{N}(0,1)$) dataset sampled from the function. Five samples are taken, each containing twenty instances. (b), (c), (d) are five polynomial fits, namely, $g_i(\cdot)$, of order 1, 3, and 5. For each case, dotted line is the average of the five fits, namely, $\overline{g}(\cdot)$.

# Polynomial Regression

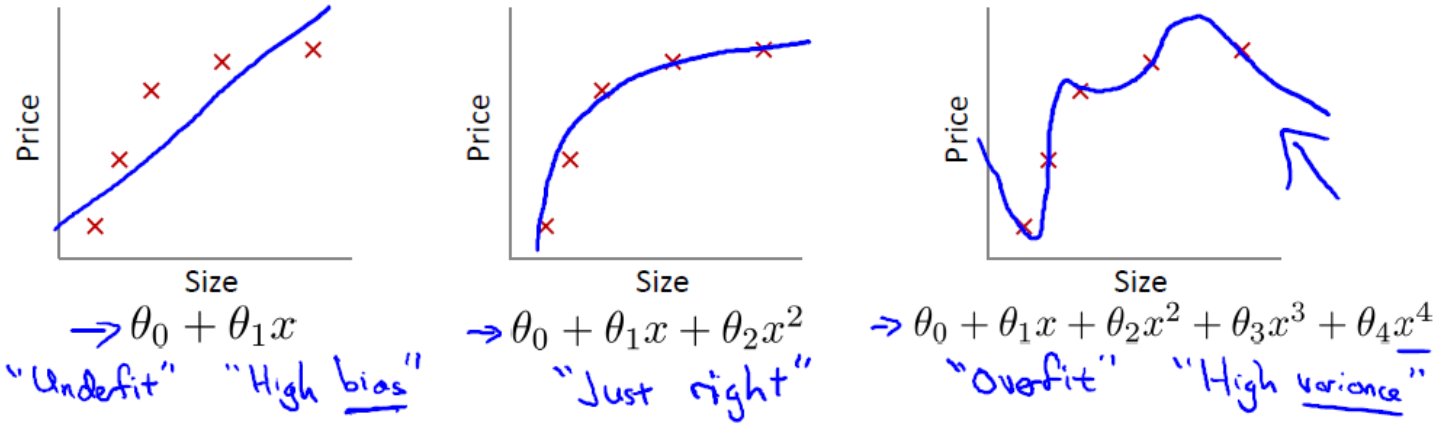



- Order 1 has the smallest variance.
- Order 5 has the smallest bias.
- As the order is increased, bias decreases but variance increases.
- Order 3 has the minimum error.

# Regularization - The problem of overfitting

**Complex model:** The learned hypothesis may fit the training set very well, but fail to generalize to new examples

Example: Linear regression (housing prices)



# Regularization

- Reduce magnitude/values of parameters .
- Works well when we have a lot of features, each of which contributes a bit to predicting y.
- Penalize complex models

  E’=error on data + λ model complexity

# Ridge Regression (L2 regularization)

- In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients

$$E(\mathbf{w} \mid \mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}\left[r^t - g\left(x^t \mid \mathbf{w}\right)\right]^2 + \lambda \sum_i w_i^2$$

- The penalty term (lambda) regularizes the coefficients such that if the coefficients take large values the optimization function is penalized. So**, ridge regression shrinks the coefficients and it helps to reduce the model complexity**

# Ridge Regression (L2 regularization)

***Example:*** In regularized linear regression

$$E(\mathbf{w}|\mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}\left[r^t - g\left(x^t|\mathbf{w}\right)\right]^2 + \lambda\sum_i w_i^2$$

What if $\lambda$ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting to be very large can't hurt it
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

# Lasso Regression (L1 regularization)

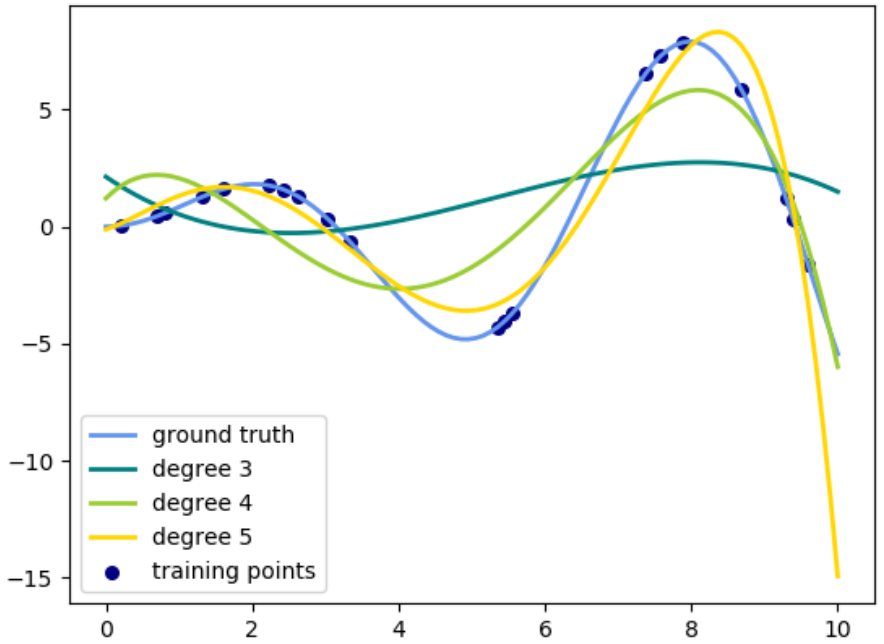
- The cost function for Lasso (least absolute shrinkage and selection operator) regression can be written as

$$E(w|X) = \frac{1}{2}\sum_{t}(r^t - g(x^t|w))^2 + \lambda \sum_{i} |w_i|$$

- *The only difference is instead of taking the square of the coefficients, magnitudes are taken into account.*
- This type of regularization (L1) can lead to zero coefficients i.e. some of the features are completely neglected for the evaluation of output.
- **So Lasso regression not only helps in reducing over-fitting but it can help us in feature selection**

# Polynomial interpolation

```
print(__doc__)

# Author: Mathieu Blondel
#         Jake Vanderplas
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline


def f(x):
    """ function to approximate by polynomial interpolation"""
    return x * np.sin(x)


# generate points used to plot
x_plot = np.linspace(0, 10, 100)

# generate points and keep a subset of them
x = np.linspace(0, 10, 100)
rng = np.random.RandomState(0)
rng.shuffle(x)
x = np.sort(x[:20])
y = f(x)

# create matrix versions of these arrays
X = x[:, np.newaxis]
X_plot = x_plot[:, np.newaxis]

colors = ['teal', 'yellowgreen', 'gold']
lw = 2
plt.plot(x_plot, f(x_plot), color='cornflowerblue', linewidth=lw,
         label="ground truth")
plt.scatter(x, y, color='navy', s=30, marker='o', label="training points")

for count, degree in enumerate([3, 4, 5]):
    model = make_pipeline(PolynomialFeatures(degree), Ridge())
    model.fit(X, y)
    y_plot = model.predict(X_plot)
    plt.plot(x_plot, y_plot, color=colors[count], linewidth=lw,
             label="degree %d" % degree)

plt.legend(loc='lower left')

plt.show()
```




https://scikit-learn.org/stable/auto_examples/linear_model/plot_polynomial_interpolation.html#sphx-glr-auto-examples-linear-model-plot-polynomial-interpolation-py