8.Linear Model Part 3: Polynomial and logistic regression, learning curves, bias-variance, regularization

M.A.Z. Chowdhury and M.A. Oehlschlaeger

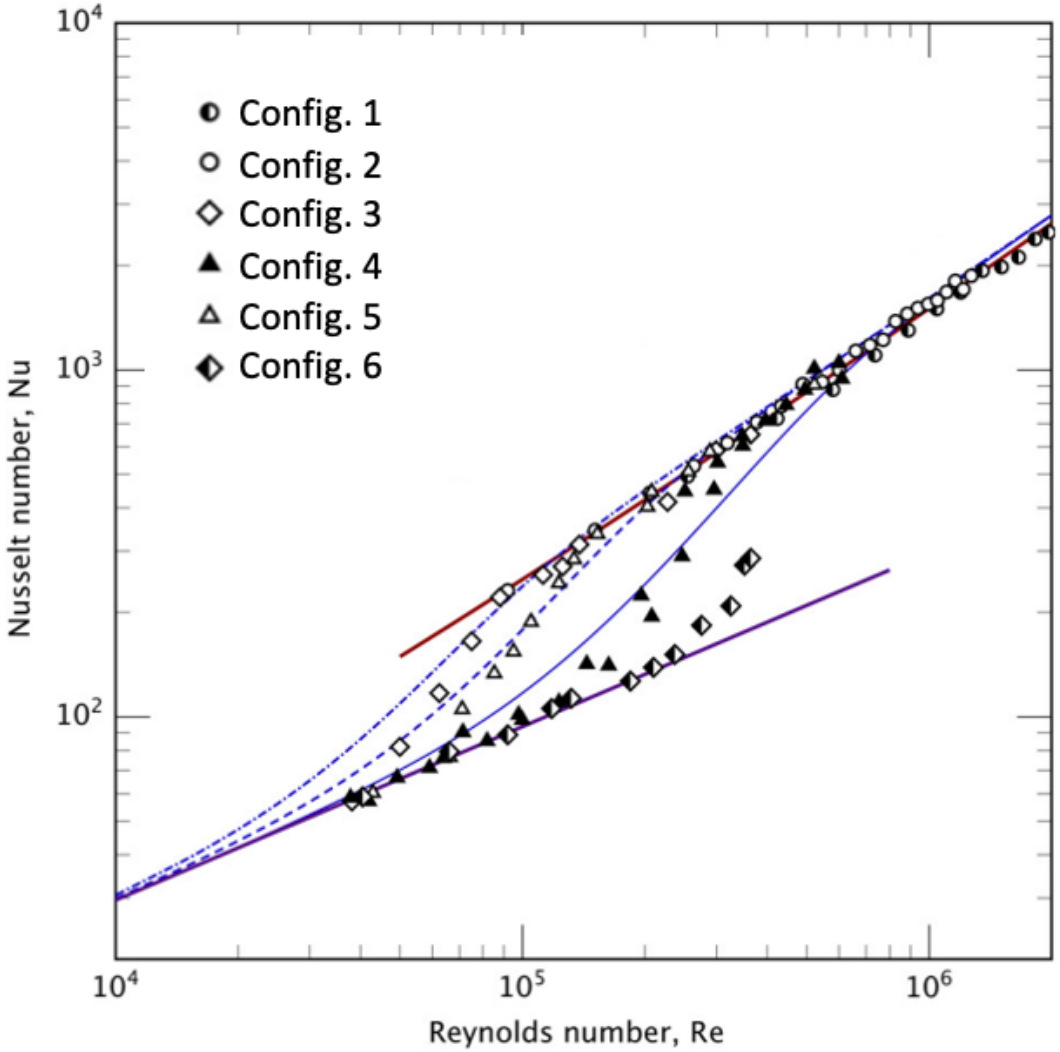
Department of Mechanical, Aerospace and Nuclear Engineering
Rensselaer Polytechnic Institute, Troy, New York

chowdm@rpi.edu
oehlsm@rpi.edu

"Amongst competing hypotheses, the simplest is the best"
- Occam's razor

MANE 4962 and 6962

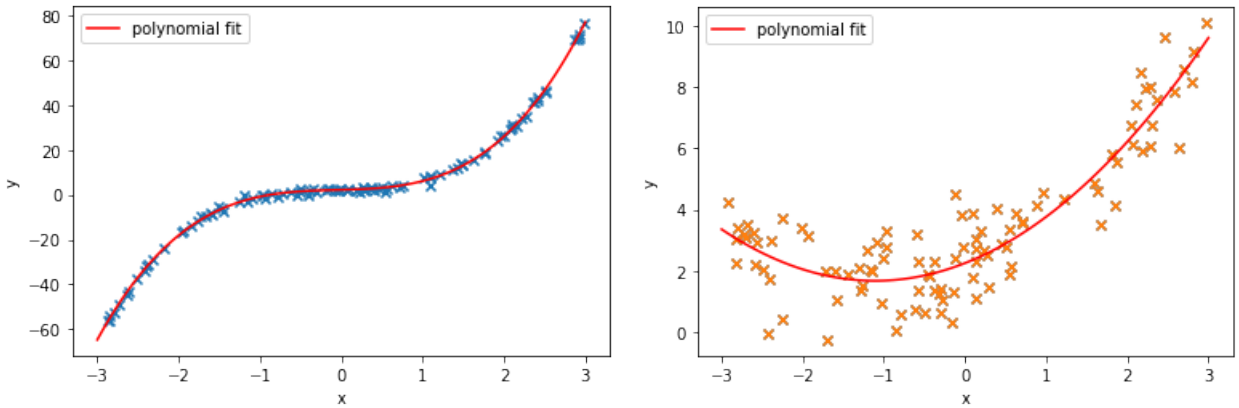
Digitizing data



https://plotdigitizer.com/app

Polynomial Regression

We can still use a linear model to fit nonlinear data. Add powers of each feature as new features, then train a linear model on extended set of features.

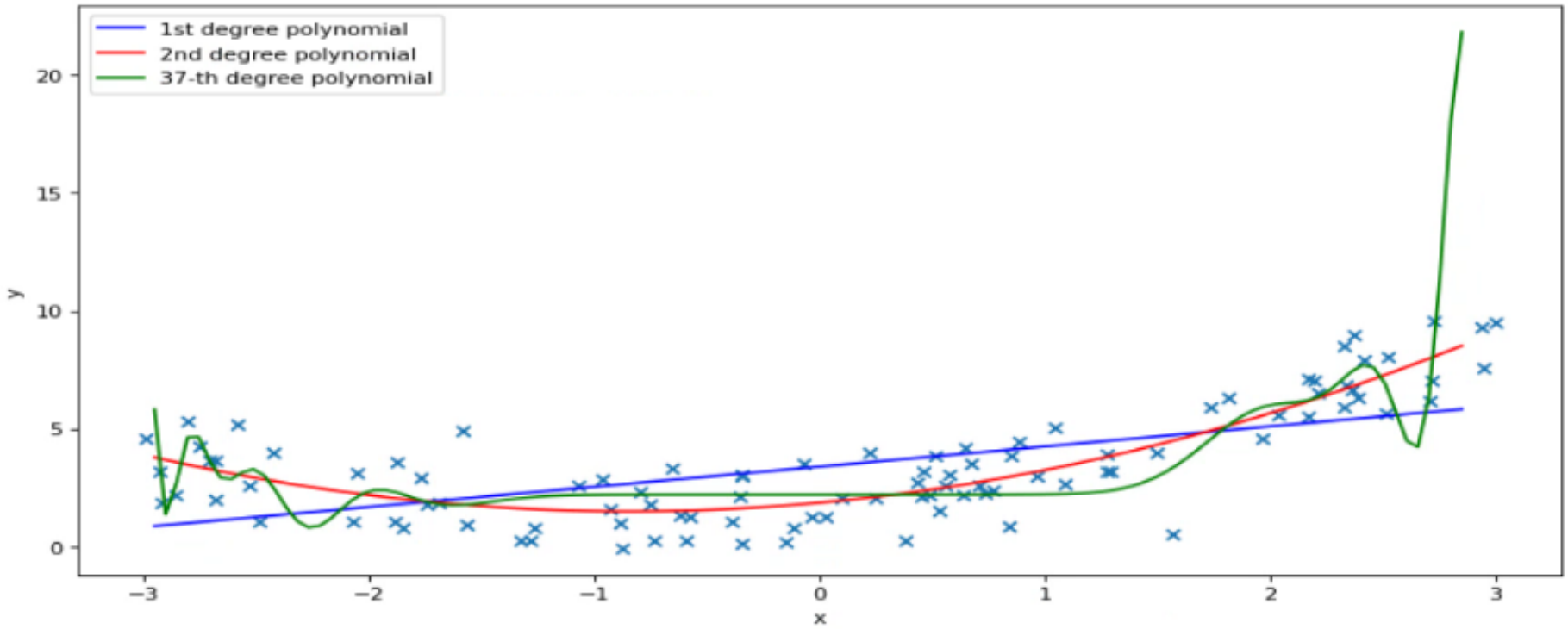



```
1   from sklearn.linear_model import LinearRegression
2   from sklearn.preprocessing import PolynomialFeatures
3   poly_features = PolynomialFeatures(degree=2, include_bias=False)
4   X_poly = poly_features.fit_transform(X)
5   lin_reg = LinearRegression()
6   lin_reg.fit(X_poly, y)
7   lin_reg.intercept_, lin_reg.coef_
```

PolynomialFeatures(degree=d) transforms an array containing n features into an array containing $\frac{(n+d)!}{d!n!}$ features, where n! is the factorial of n, equal to $1 \times 2 \times 3 \times \ldots \times n$. Beware of the combinatorial explosion of the number of features!

Overfitting and underfitting

A high degree polynomial will wiggle and regress much better than linear regression.
The 37-th degree Polynomial Regression model is severely overfitting the training data, while the linear model is underfitting it.



The quadratic model generalizes well since the data was generated using a quadratic model.

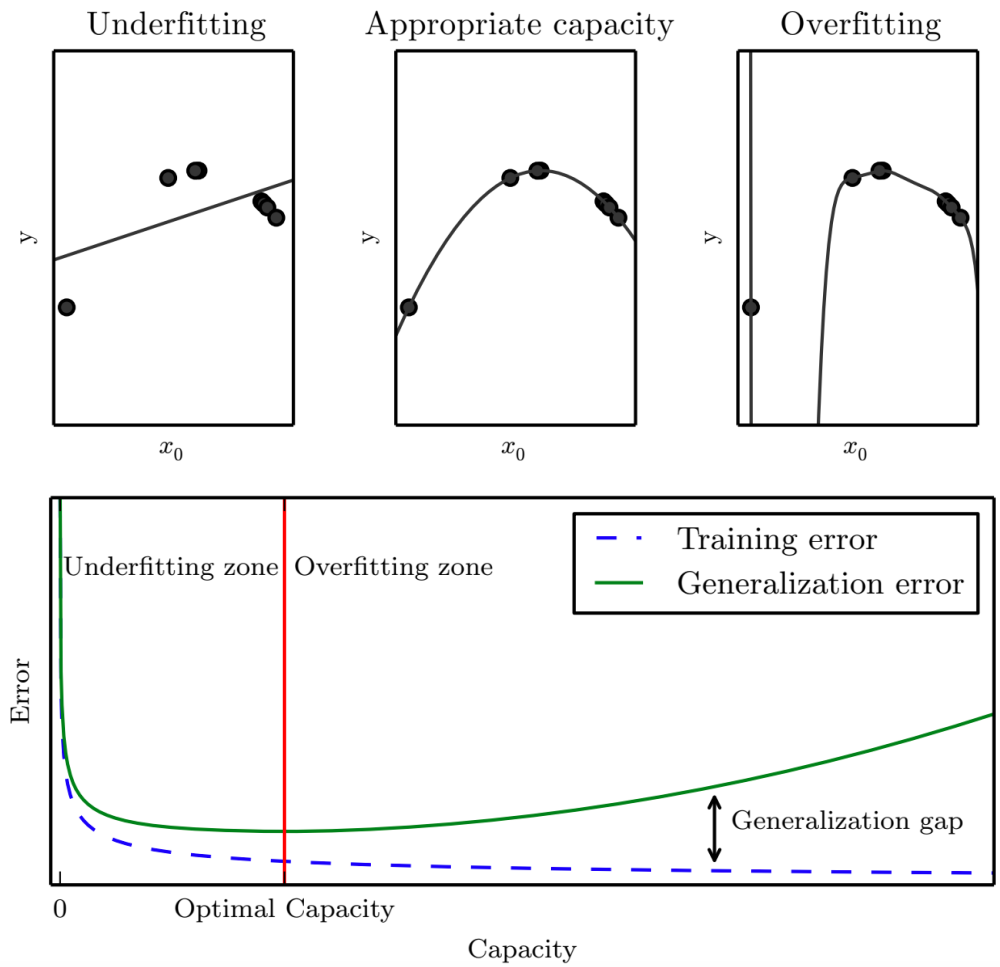
Overfitting and underfitting

Generally, we won't know the underlying model that produced the data.

If a model performs well on the training data but generalizes poorly according to the cross-validation metrics, then your model is overfitting.

If it performs poorly on both, then it is underfitting. This is one way to tell when a model is too simple or too complex.

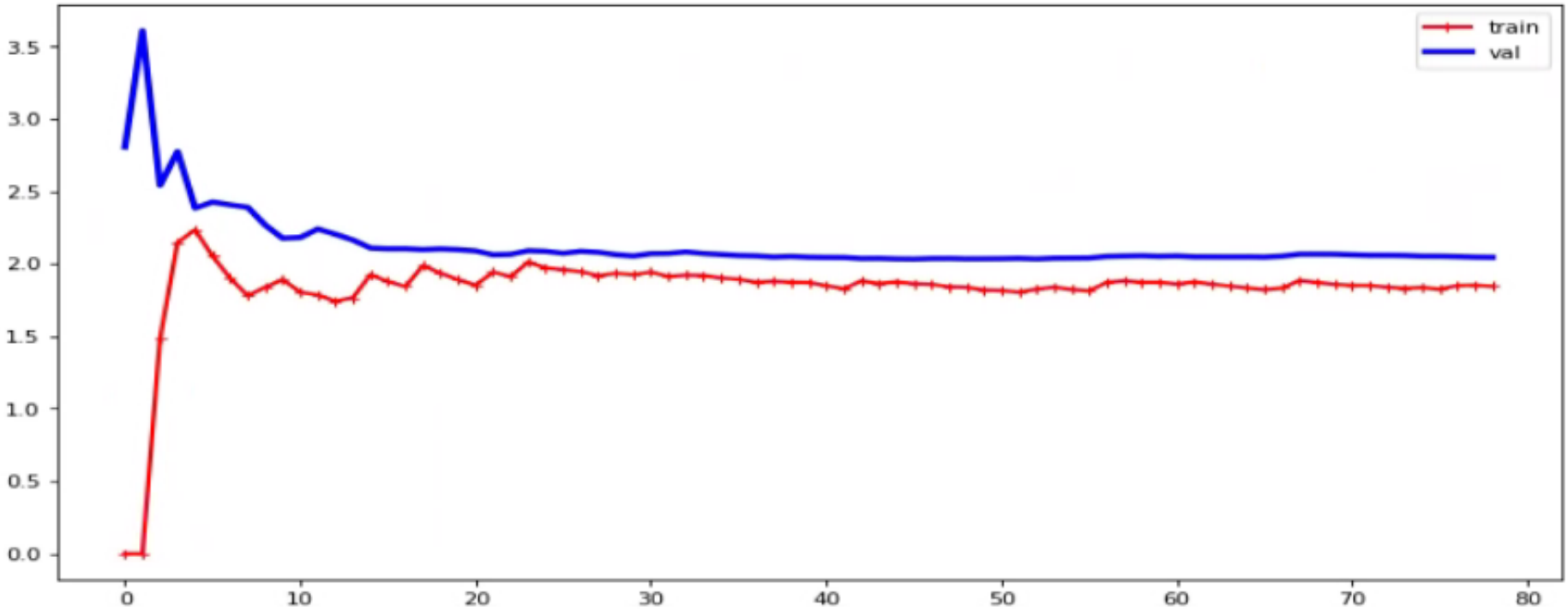
Capacity



Capacity controls whether a model is more likely to overfit or underfit. A model's capacity is its ability to fit a wide variety of functions.

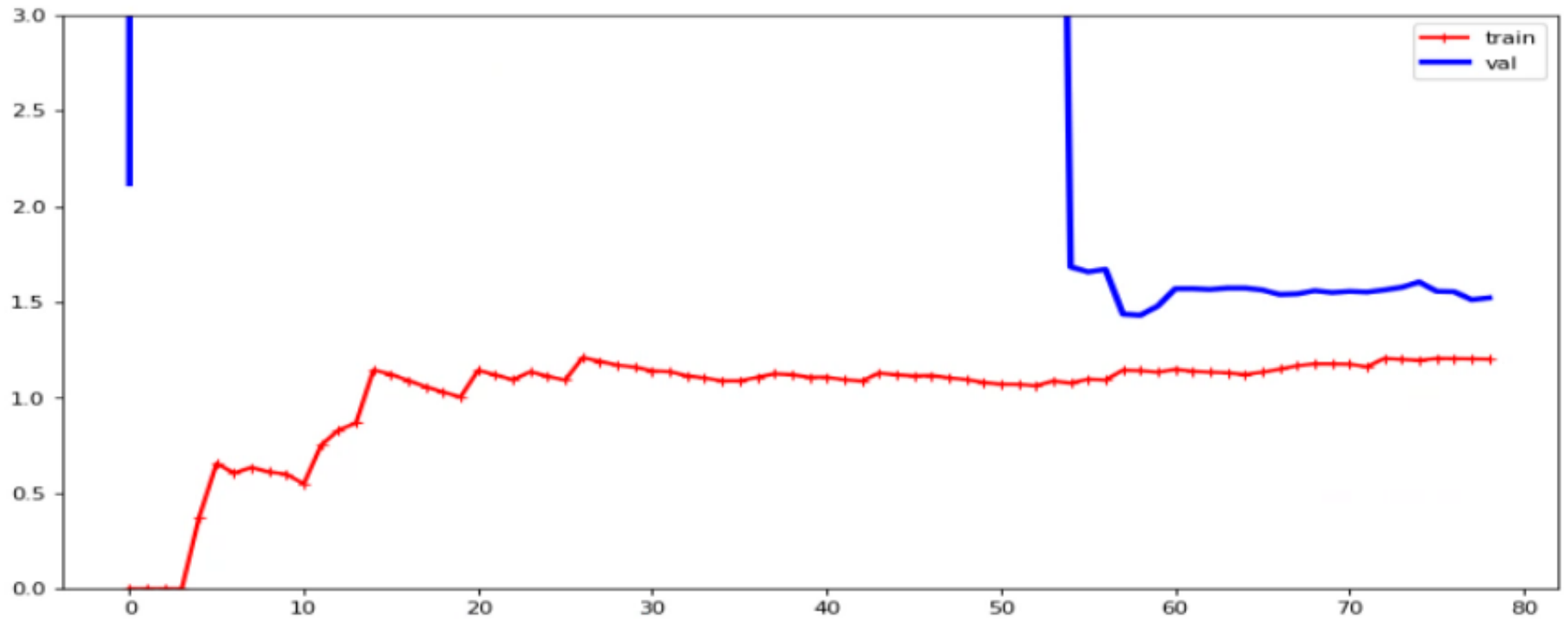
Learning Curves for the linear model

Learning curves are plots of the model's performance on the training set and the validation set as a function of the training set size or the training iteration.
To generate the plots, train the model several times on different sized subsets of the training set.



- The linear model is underfitting since it can only fit few training data points well and then settles into a constant error due to being incapable of fitting complex data.
- It is also unable to reduce the error close to zero on the validation data.

Learning Curves for the 37-th degree model



- The error on the training data is much lower compared to the linear model.
- But there is a gap between the errors on training and validation data.
- If you use more data then the errors will approximate each other and reduce overfitting.

Bias-variance trade-off

- An important theoretical result of statistics and Machine Learning is the fact that a model's generalization error can be expressed as the sum of three very different errors.

Bias

Part of the generalization error due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A high-bias model is most likely to underfit the training data.

Variance

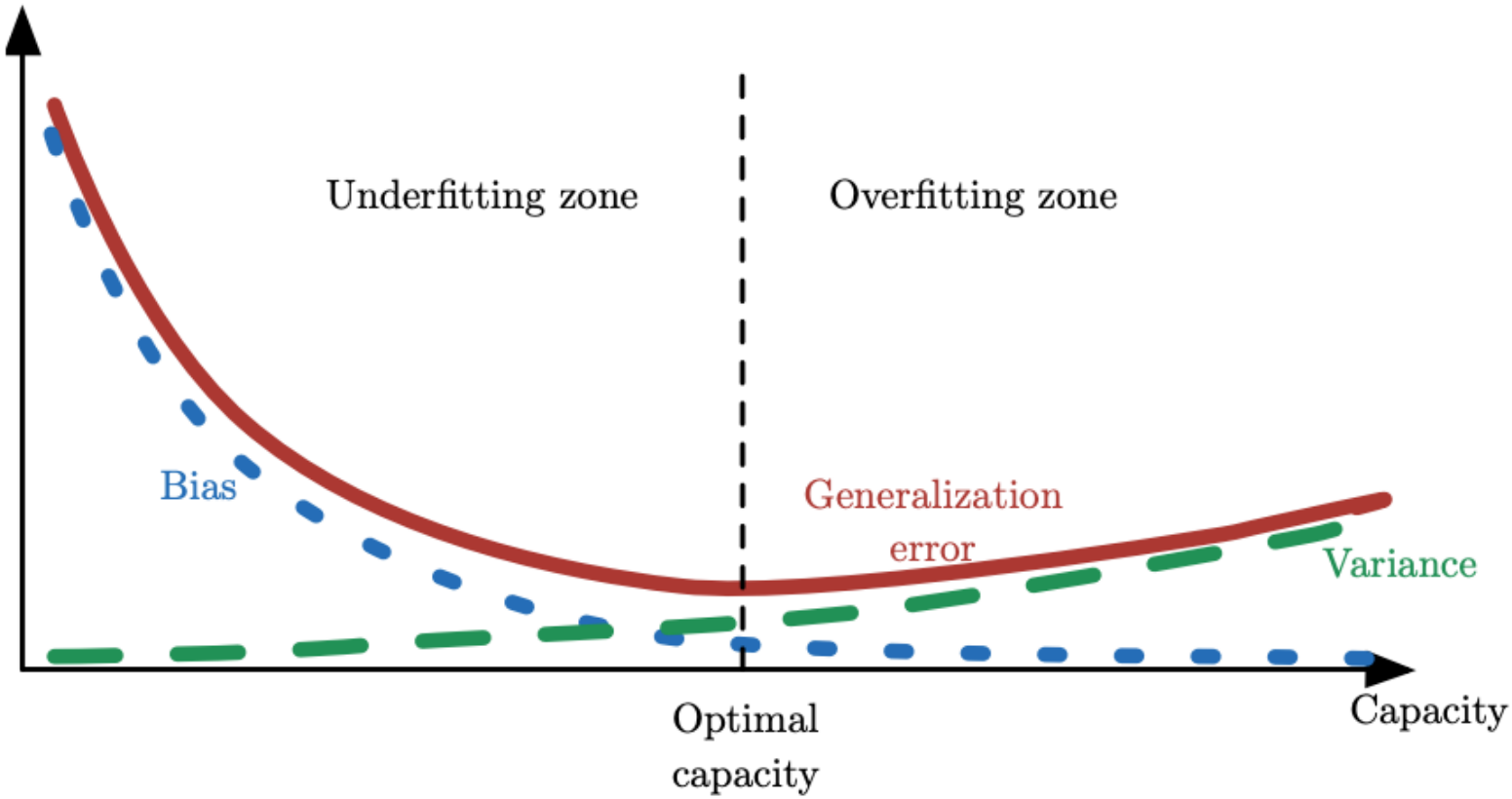
Due to the model's excessive sensitivity to small variations in the training data. A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance and thus overfit the training data.

Irreducible error

Due to the noisiness of the data itself. The only way to reduce this part of the error is to clean up the data (e.g., fix the data sources, such as broken sensors, or detect and remove outliers).

- Increasing a model's complexity (capacity) will typically increase its variance and reduce its bias. Conversely, reducing a model's complexity increases its bias and reduces its variance. This is why it is called a trade-off.

Bias-variance trade-off



Regularized linear models

- To regularize a model, means to constrain it and not make it too complex.

Ridge

Will force model to choose smaller weights. $\lambda \sum |w_i|^2$ term is added to cost function. λ is the strength of regularization, a hyperparameter.

Lasso

Make weights of less important features zero. $\lambda \sum |w_i|$. Lasso Regression automatically performs feature selection and outputs a sparse model (i.e., with few nonzero feature weights).

ElasticNet

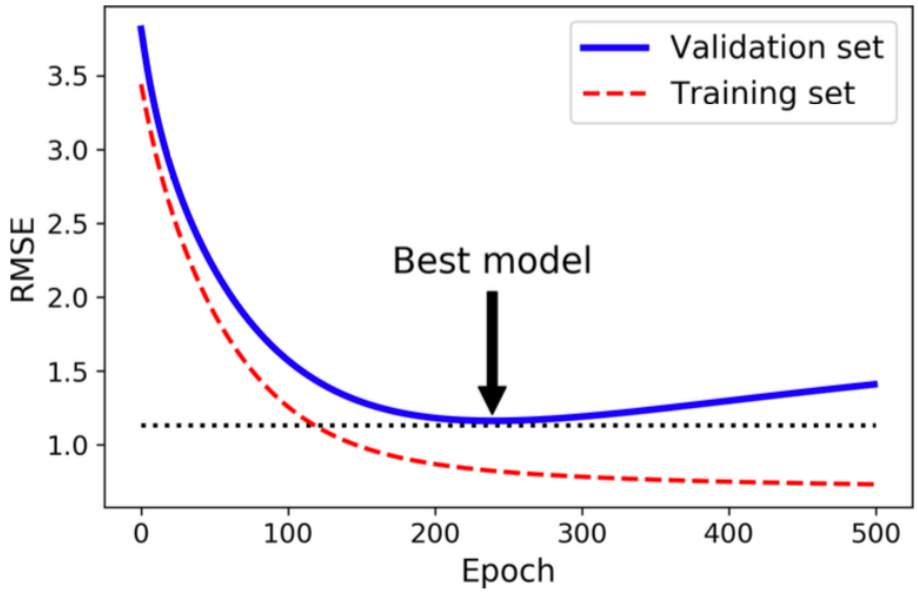
Combines both L1 and L2. $r\lambda \sum |w_i| + \frac{1-r}{2}\lambda \sum |w_i|^2$ term is added to cost function, r is the mix ratio.

Regularized linear models

- The gradients get smaller as the parameters approach the global optimum, so Gradient Descent naturally slows down, which helps convergence (as there is no bouncing around).
- The optimal parameters get closer and closer to the origin when you increase α, but they never get eliminated entirely.
- Gradually reduce the learning rate during training so there will not be any bouncing when GD converge.
- Lasso cost function is not differentiable at $w_0 = 0$.

Regularization via early stopping

Stop training as soon as the validation error reaches a minimum.



- When the validation error stops decreasing and starts to go back up it means the model has started to overfit the training data.
- Stochastic and Mini-batch Gradient Descent curves are not smooth, and it may be hard to know whether you have reached the minimum or not.
- One solution is to stop only after the validation error has been above the minimum for some time when you are confident that the model will not do any better.
- Then roll back the model parameters to the point where the validation error was at a minimum.

Logistic Regression

Consider, m data points. We want to predict y from a input variables or features x_1, x_2

example	input $\underline{x} = (x_1, x_2)$	output y	prediction $\hat{y}$
1	$x_1^{(1)}, x_2^{(1)}$	y_1 = 0 or 1	$\hat{y_1} = [0, 1]$
2	$x_1^{(2)}, x_2^{(2)}$	y_2 = 0 or 1	$\hat{y_2} = [0, 1]$
3	$\vdots$	$\vdots$	$\vdots$
m	$x_1^{(m)}, x_2^{(m)}$	y_m = 0 or 1	$\hat{y_m} = [0, 1]$

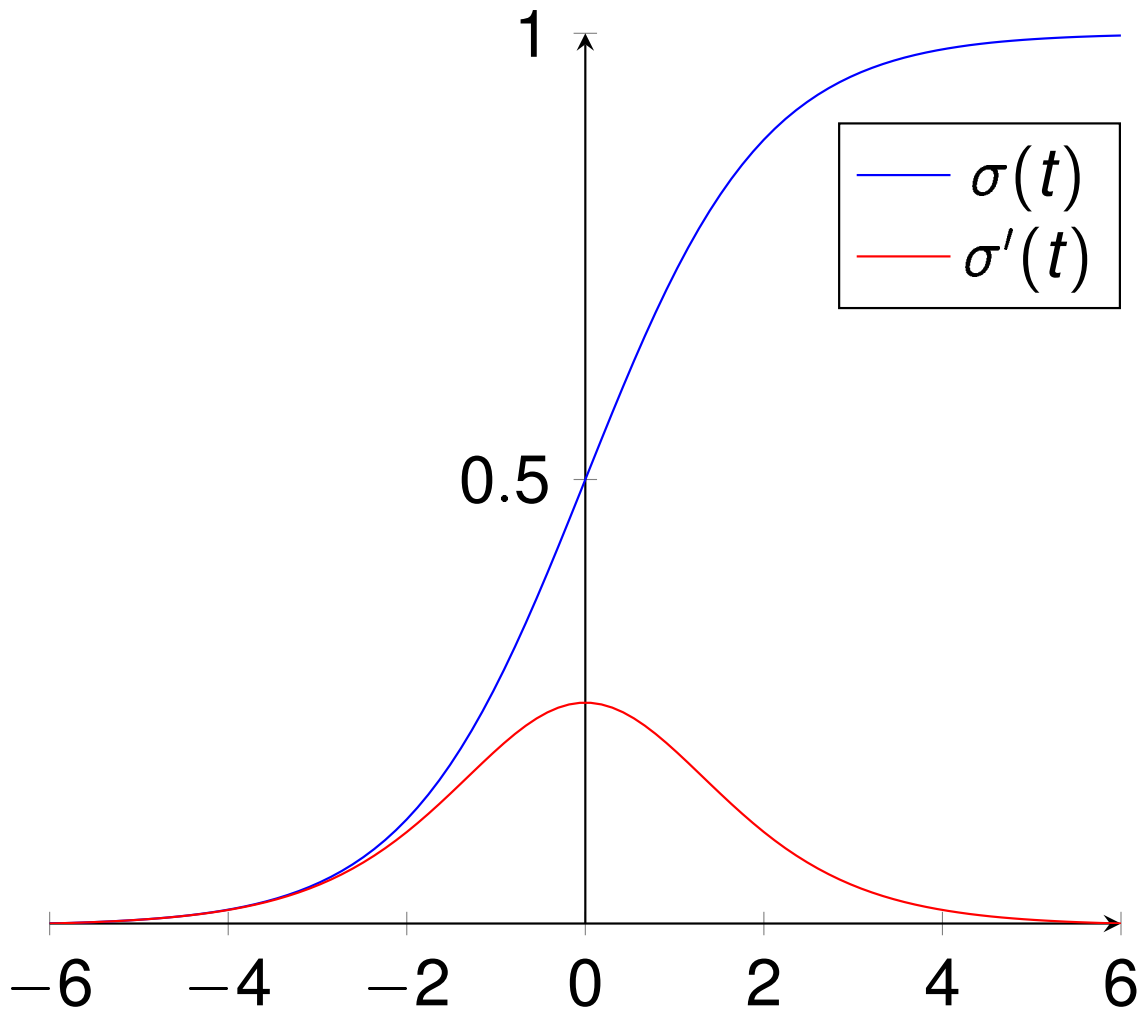
Logistic Regression

- $\hat{p} = h(\underline{x}; \underline{w}) = \sigma(w_0 + w_1 x_1 + + w_2 x_2)$
- $z = w_0 + w_1 x_1 + + w_2 x_2$ (say)
- $\sigma(t) = \frac{1}{1+e^{-t}}$
- $\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$
- Logistic Regression (also called Logit Regression) is used to estimate the "probability" that an example belongs to a particular class.
- If the estimated probability is greater than 50%, then the model predicts that the example belongs to that class (called the positive class, labeled “1”), and otherwise it predicts that it does not belong to the negative class, labeled “0”. This makes it a binary classifier.
- It handles extreme feature values quite well.

Logistic Regression

- The sigmoid or logistic function squishes the inputs to [0,1].
- $\sigma(t) = \frac{1}{1+e^{-t}}$
- $\lim_{t \to \infty} \sigma(t) = 1$
- $\lim_{t \to -\infty} \sigma(t) = 0$
- $\sigma(t = 0) = \frac{1}{2} = 0.5$
- The score t is often called the logit.
- The logit function, $logit(p) = log(\frac{p}{1-p})$, is the inverse of the logistic function.
- The logit of the estimated probability p, is t.
- The logit is also called the log-odds, since it is the log of the ratio between the estimated probability for the positive class and the estimated probability for the negative class.

Sigmoid function



$\sigma'(t) = \sigma(t)[1 - \sigma(t)]$ [Check it]

Cost function

How to train a logistic model for classification?

$\rightarrow$ "Squish" outputs between 0 and 1

$\rightarrow$ Binary classification

Least squares or measuring absolute or squared error is not a good cost function for classification because it does not penalize enough. So we use log-loss cost function.

- cost function, $J = -\frac{1}{m}\sum_i [y_i ln\hat{y}_i + (1 - y_i) ln(1 - \hat{y}_i)]$
- If $y_i = \hat{y}_i$, for all data points then $J = 0$, the best case scenario.
- If $y_i \neq \hat{y}_i$, for some data points then $J \rightarrow \infty$, the best case scenario.
- J should be positive, differentiable, and continuous.
- $y_i ln\hat{y}_i$ and $(1 - y_i) ln(1 - \hat{y}_i)$ both terms are negative since the output of the model is either 0 or 1.

Outputs of this cost function

Let's consider a single data point,

- $J = -[y ln \hat{y} + (1 - y) ln(1 - \hat{y})]$

Correct classification

- $J \approx 0$, when $y = 0$ and prediction is $\hat{y} \approx 0$
- $J \approx 0$, when $y = 1$ and prediction is $\hat{y} \approx 1$

Incorrect classification

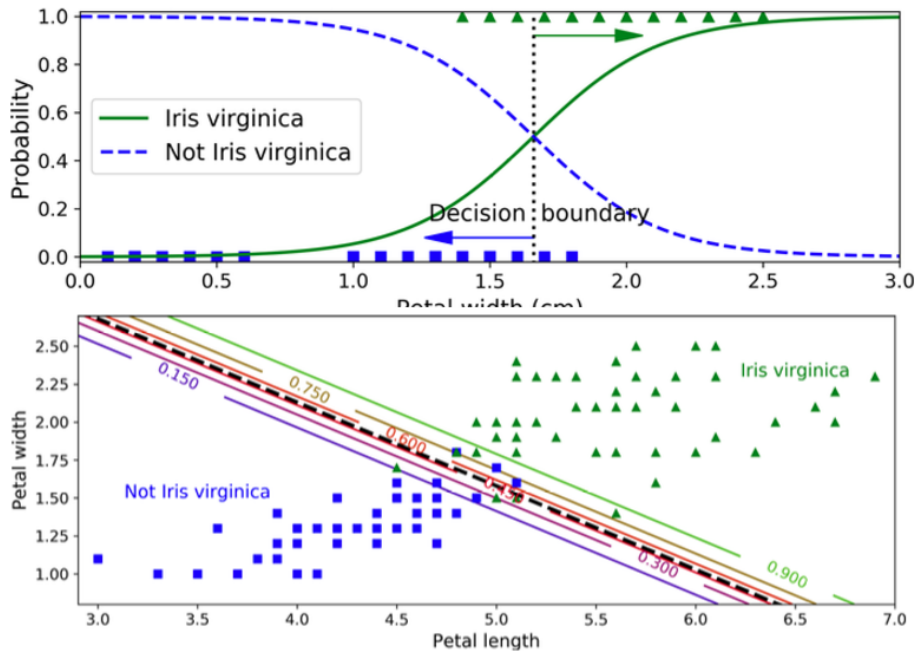
- $J \to \infty$, when $y = 0$ and prediction is $\hat{y} \approx 1$
- $J \to \infty$, when $y = 1$ and prediction is $\hat{y} \approx 0$

Comments on this cost function

- The least square error cost function would not be as high for the incorrect classifications.
- However, there is no closed-form solution of this cost function.
- The partial derivative have similar form: $\frac{\partial J}{\partial w_j} = \frac{1}{m}\sum_i(\hat{y}_i - y_i)x_j$
- Note: Scikit by default will apply L2 regularization to the logistic regression model.
- Scikit also uses inverse of the regularization strength.

Analyzing the classification model

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
iris = datasets.load_iris()
print(list(iris.keys()))
X = iris["data"][:, 3:]
y = (iris["target"] == 2).astype(np.int)
log_reg = LogisticRegression()
log_reg.fit(X, y)
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-",
label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--",
label="Not Iris virginica")
plt.xlabel('petal width (cm)')
plt.ylabel('probability')
plt.legend()
```



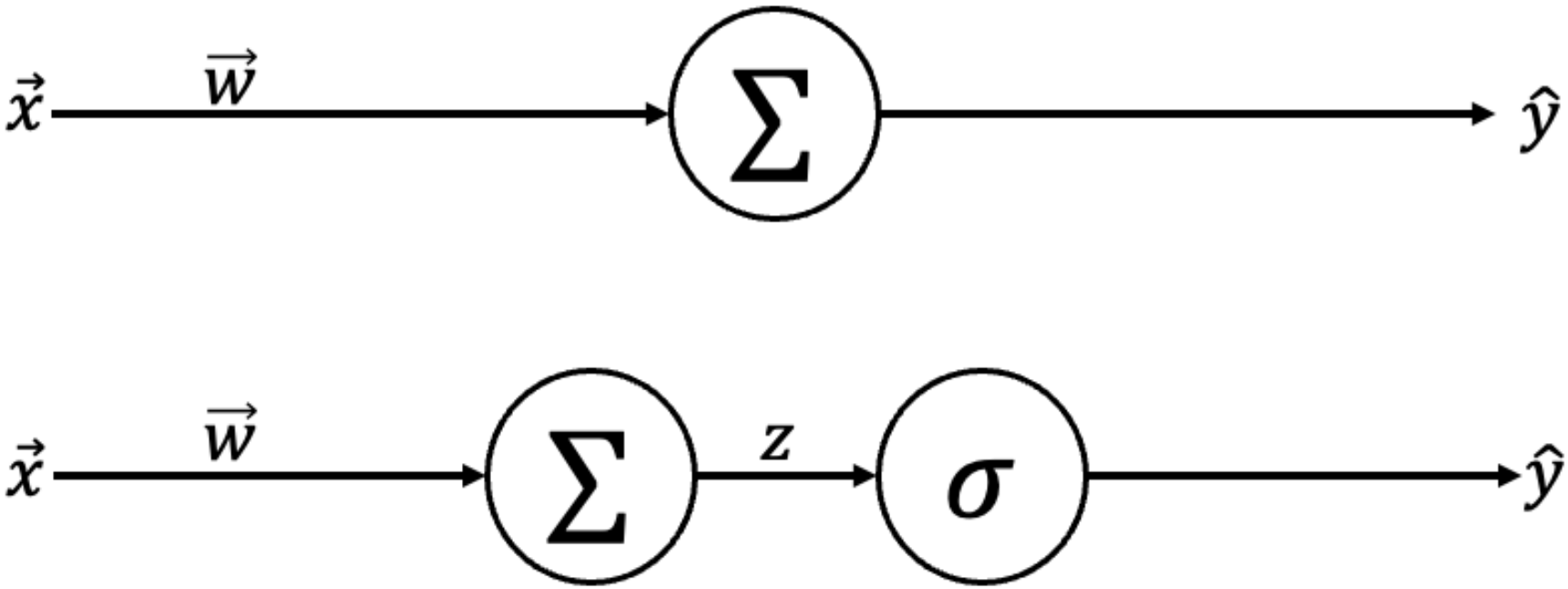
☞ The decision boundary is linear for logistic regression classification models.

Aurélien Géron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

Recursive Feature Elimination

- For a machine learning task not all features may be useful.
- Progressively select a small subset of features via cross-validation.

```
1  import numpy as np
2  from sklearn import datasets
3  from sklearn.model_selection import train_test_split
4  from sklearn.feature_selection import RFE
5  from sklearn.linear_model import LogisticRegression
6  import matplotlib.pyplot as plt
7  iris = datasets.load_iris()
8  X = iris["data"]
9  y = iris["target"]
10 print('original features and their names : ', iris.feature_names)
11 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
12 lr = LogisticRegression(solver='lbfgs', max_iter=1000)
13 rfe = RFE(lr, n_features_to_select=2)
14 rfe = rfe.fit(X_train, y_train)
15 print('True/False Selection of features:', rfe.support_)
16 feature_indices = np.array([0,1,2,3])
17 feature_indices = feature_indices[rfe.support_]
18 plt.scatter(X_train[:,feature_indices[0]], X_train[:,feature_indices[1]], c =y_train)
19 plt.xlabel(iris.feature_names[feature_indices[0]])
20 plt.ylabel(iris.feature_names[feature_indices[1]])
21 print('The best features are : ', np.array(iris.feature_names)[rfe.support_])
```


$\vec{x}$
$\vec{w}$
Σ
$\hat{y}$
$\vec{x}$
$\vec{w}$
Σ
z
σ
$\hat{y}$