

# MSc Future Power Networks Smart Grid Technology (ELEC97077) Machine Learning for Power System A Closer Look

Dr. Wangkun Xu  
wangkun.xu18@imperial.ac.uk

Department of EEE, Imperial College London

Oct. 31, 2024 (last modified: Oct. 30, 2024)



# Outline I

## Introduction

- Machine Learning: Why and What
- Copyright and Reference
- Coursework Overview

## Supervised Learning

- Overview
- Linear Regression
- Logistic Regression
- Support Vector Machine

## Unsupervised Learning

- Overview
- Principal Component Analysis
- k-means Clustering

## Reinforcement Learning

- Overview Only

## Performance Measure

- Overview



# Outline II

## Model Capacity

Underfit and Overfit

Regularization

## Hyperparameters and Validation Set

Hyperparameters

Validation Set



# Outline

## Introduction

Machine Learning: Why and What

Copyright and Reference

Coursework Overview



# Machine Learning: Why and What I

From the previous lectures, we learn that the new trend of large penetration of **renewable energy** challenges the old model-based control room techniques.

- ▶ The grid with power electronics becomes a **black box** to the system operator.
- ▶ Compared to traditional energy resources, renewable energy is more difficult to manage.

We are entering the era of **big data**. As the advances of **digitalization** in power system, more and more data from the measurement unit, such as remote terminal units (RTUs), phasor measurement units (PMUs) and smart meters, are readily available.



# Machine Learning: Why and What II



This requires automated data analysis methods, which is what *machine learning* (ML) provides.




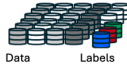
## Definition

In particular, we define machine learning as a set of methods that can automatically detect patterns in data and then use the uncovered patterns to predict future data or to perform other types of decision-making under uncertainty.



# Machine Learning: Why and What III

Machine and deep learning is a very broad and practical topic. Therefore, this lecture only covers the most basic and ready-to-use ones in the power system. Many of the ideas can be used for more advanced topics.

	Expert Systems	Machine Learning	Deep Learning	Foundation Model
	 Domain Knowledge Rules	 Data Labels Domain Knowledge	 Data Labels	 Data Labels
<b>Model Size</b>	Very small	Small	Large	Very Large
<b>Features</b>	Hand-designed	Hand-designed	Learnt	Learnt
<b>Learning</b>	None	Supervised	Supervised	Self-supervised
<b>Data</b>	Very few	Few labeled Data	Labeled data	Very large unlabeled + small labeled data
<b>Adaptability</b>	None	Little	Medium	Large

**Figure:** The evolution of AI (Source: [Hamann et al., 2024]). This course mainly focuses on the **machine learning** part. Fundamental ideas are shared for deep learning and foundation model.



# Machine Learning: Why and What IV

A little bit formal definition:

## Definition

A computer program is said to learn from experience  $E$  with respect to some class of task  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .

Based on the definition, machine learning algorithms can be classified according to different tasks ( $T$ ), experience ( $E$ ), and performances ( $P$ ),

- ▶ Based on the learning task, we have
  - ▶ *Regression Task*: the computer program is asked to predict a **numerical** value given some input. To solve this task, the learning algorithm is asked to output a function  $f: \mathbb{R}^D \rightarrow \mathbb{R}'$ .
  - ▶ *Classification Task*: the computer program is asked to specify which of  $k$  categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function  $f: \mathbb{R}^D \rightarrow \{1, \dots, k\}$ . Sometimes,  $f$  can produce a **probability distribution** over classes.

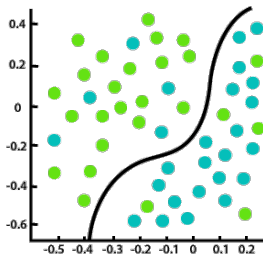


# Machine Learning: Why and What V

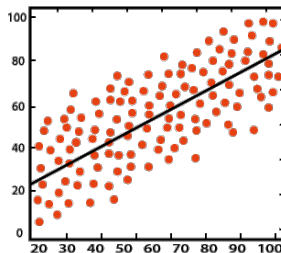
- ▶ Based on the kind of experience the data can have during training, we have
  - ▶ Supervised Learning
  - ▶ Unsupervised Learning
  - ▶ Reinforcement Learning
- ▶ In order to evaluate the abilities of a machine learning algorithm, we must design various quantitative measures of its performance.



# Machine Learning: Why and What VI



Classification



Regression

Figure: Regression vs classification. Source:

<https://www.javatpoint.com/regression-vs-classification-in-machine-learning>



# Machine Learning: Why and What VII

## Question

Determine the tasks of the following activities in power system:

1. Load forecast.
2. Detect false data injection attack.
3. Predict the optimal dispatch plan of generator.



# Outline

## Introduction

Machine Learning: Why and What

Copyright and Reference

Coursework Overview



## Copyright and Reference I

Some of the materials (including images and mathematical definitions) in this lecture note are taken from the following references:

- ▶ Murphy, K. P. Machine Learning—A probabilistic Perspective. The MIT Press, 2012.
- ▶ Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning. The MIT Press, 2016.
- ▶ Bishop, Christopher M., and Nasser M. Nasrabadi. Pattern recognition and machine learning. Vol. 4. No. 4. New York: springer, 2006.

You are welcome (but not compulsory) to self-study online for various topics. For example,

- ▶ Machine learning specialization by Andrew Ng<sup>1</sup>.

---

<sup>1</sup>https:

//www.youtube.com/playlist?list=PLkDaE6sCZn6FNC6YRfRQc\_FbeQrF8BwGI ▶



# Outline

## Introduction

Machine Learning: Why and What

Copyright and Reference

Coursework Overview



# Coursework Overview I

In the previous lectures, you have already learned some ML applications for the power system and become familiar with many tasks of the power system. This lecture will take a closer look on the details of machine learning and how to build a successful ML pipeline for power system tasks (e.g. the coursework!).

The coursework **may** include (you will be assigned the coursework later)

- ▶ Regression task: predict the output of DC-OPF.
- ▶ Classification task: detect on a false data injection attack.

Note that the above topic may change, but the ML techniques are the same and will be covered in this lecture.



## Coursework Overview II

The coursework will include both mathematical questions and programming tasks. Some questions are from the lecture note. The programming tasks will be run as a competition. I.e., each group (two students) should upload their best forecast results, and I will run the grading algorithm for the final score.

You may use any programming language, for example Python or Matlab, for the coursework. However, data will be allocated in Python `.mat` (numpy) format only.

You are not asked to implement ML algorithm from beginning. Instead, use well-developed function in Matlab machine learning toolbox or Python sklearn.



# Outline

## Supervised Learning

### Overview

Linear Regression

Logistic Regression

Support Vector Machine



# Overview I

## Definition

In *supervised learning* approach, the goal is to learn a mapping from inputs  $x$  to output  $y$ , given a **labeled set of input-output pairs**  $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ . Here  $\mathcal{D}$  is the *training set*, and  $N$  is the number of training examples.

Each  $x_i$  is a  $D$ -dimensional vector of numbers, representing, say, the power injection and flow measurements from the RTUs. These are called *features* or *attributes*. They are stored in an  $N \times D$  *design matrix*  $X$ .

$$X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix} = \begin{pmatrix} x_1^1 & x_1^2 & \cdots & x_1^D \\ x_2^1 & x_2^2 & \cdots & x_2^D \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^D \end{pmatrix} \in \mathbb{R}^{N \times D}$$

In principal,  $x_i$  can contain complex information, such as an image, sentence, etc. In this lecture, we mainly focus on **time series** data.



# Overview II

Depending on the format of the label  $y$ , we have

- ▶  $y_i$  is real-valued: *regression* or *pattern recognition* task.
- ▶  $y_i$  is categorical: *classification*.



# Outline

## Supervised Learning

Overview

**Linear Regression**

Logistic Regression

Support Vector Machine



# Linear Regression: Scalar Case I

One of the most widely used models for regression is known as *linear regression*. This asserts that the response is a linear function of the inputs.

$$y(x) = w^T x + \epsilon = \sum_{j=1}^D w_j x_j + \epsilon$$

where  $D$  is the number of features and  $\epsilon$  is the residual error between the linear predictions and the true response.  $y$  is a **scalar**.  $w$  is the model parameter that you want to learn.

We often assume that  $\epsilon$  has a Gaussian distribution. e.g.,  $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$ . Then the linear regression can be denoted as

$$p(y|x, \theta) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$



## Linear Regression: Scalar Case II

In this case, it is assumed that  $y$  follows a conditional Gaussian distribution with mean given by

$$\mu(x) = w^T x$$

and the variance equals to the **fixed** noise  $\sigma^2(x) = \sigma^2$ .

For example, suppose the input is one-dimensional,

$$\mu(x) = w_0 + w_1 x = w^T x$$

where  $w_0$  is the *bias* term,  $w_1$  is the slope, and we can defined the vector  $x = (1, x)$ .



## Linear Regression: Scalar Case III

The linear regression can capture nonlinear relationship between  $x$  and  $y$ .

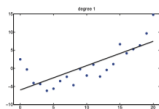
$$p(y|x, \theta) = \mathcal{N}(y|w^T \phi(x), \sigma^2)$$

where  $\phi(x)$  is a **nonlinear** function of  $x$ . For example, a *polynomial regression* is defined as

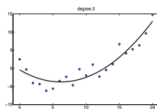
$$\phi(x) = [1, x, x^2, \dots, x^d]$$



## Linear Regression: Scalar Case IV

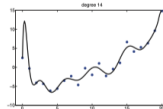


(a)

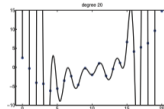


(b)

**Figure:** Linear regression (a) vs nonlinear regression (b).



(a)

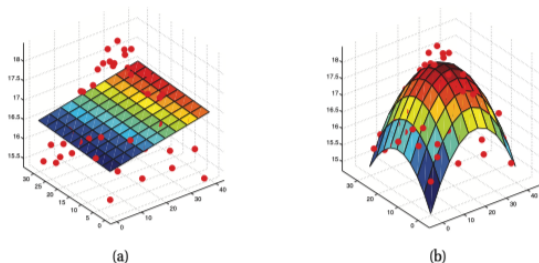


(b)

**Figure:** Polynomial of degrees 14 (a) and 20 (b).



# Linear Regression: Scalar Case V



**Figure:** Linear regression on 2D data. (a).  $\mu(x) = w_0 + w_1x_1 + w_2x_2$ ;  
(b).  $\mu(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2$



# Linear Regression: Maximum Likelihood Estimation I

Linear regression can be modelled and trained by *maximum likelihood estimation* (MLE).

$$\hat{\theta} = \arg \max_{\theta} \log p(\mathcal{D}|\theta)$$

Assuming that the training examples are independent and identically distributed (iid), the log-likelihood is represented as

$$\log p(\mathcal{D}|\theta) = \log \prod_{i=1}^N p(y_i|x_i, \theta) = \sum_{i=1}^N \log p(y_i|x_i, \theta)$$

Recall that a Gaussian distribution is assumed:

$$p(y|x, \theta) = (2\pi\sigma^2)^{-\frac{1}{2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Plugging into the log likelihood:

$$\ell(\theta) = -\frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - w^T x_i)^2 - \frac{N}{2} \log 2\pi\sigma^2$$



## Linear Regression: Maximum Likelihood Estimation II

In the case when  $\sigma$  is fixed, the second terms is removed. The so-called *mean squared error* (MSE) loss is derived:

$$\min_w \text{MSE}(w) := \sum_{i=1}^N (y_i - w^T x_i)^2$$

which is also named as *least squares*.

### Question

Can you derive the MSE loss by yourself?

The MSE loss becomes:

$$\text{MSE}(w) = \frac{1}{2} \|y - Xw\|_2^2 = \frac{1}{2} (y - Xw)^T (y - Xw)$$

where  $y \in \mathbb{R}^N$ ,  $X \in \mathbb{R}^{N \times D}$ ,  $w \in \mathbb{R}^D$ .



# Linear Regression: Maximum Likelihood Estimation III

Taking the derivative with respect to  $w$  results in the first-order optimality condition:

$$X^T y - X^T X w = 0$$

Therefore, the least-square solution is given as:

$$\hat{w}_{ls} = (X^T X)^{-1} X^T y$$



# Outline

## Supervised Learning

Overview

Linear Regression

**Logistic Regression**

Support Vector Machine



# Logistic Regression I

*Logistic regression* is a **binary classification** (though it is named as regression)...

Let's say  $y \in \{0, 1\}$ . The output of the logistic regression is the **probability** of one of the class. Therefore, we need

$$\mu(x) = \text{sigm}(w^T x) \in [0, 1]$$

A *sigmoid function* or *logistic function* is defined

$$\text{sigm}(\eta) = \frac{1}{1 + \exp(-\eta)} = \frac{e^\eta}{e^\eta + 1}$$

For the scalar case, let's assume that the output is the probability of class 1,

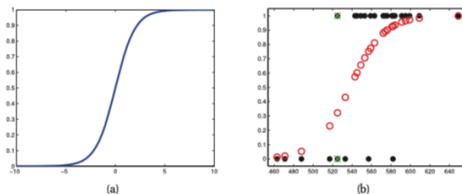
$$p(y_i = 1 | x_i, w) = \text{sigm}(w_0 + w_1 x)$$



## Logistic Regression II

We say a sample is classified as 1 if it has probability larger than 0.5:

$$\hat{y} = 1 \Leftrightarrow p(y = 1|x) > 0.5$$



**Figure:** (a). The sigmoid function; (b). The output of the logistic regression and classification outcomes. The black dots are the input data. The red circles are the predicted probabilities.



## Logistic Regression III

The log-likelihood of logistic regression is *cross-entropy* loss:

$$\text{CE}(w) = - \sum_{i=1}^N y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)$$

where  $y_i \in \{0, 1\}$ .  $\mu_i$  is the probability of class 1. Therefore, when  $y_i = 1$ , only the first-term is active (the probability of class 1); and when  $y_i = 0$ , only the second-term is active (the probability of class 0).

Unlike linear regression (MSE loss), cross-entropy loss does not have a closed-form solution. Numerical optimization techniques, such as *gradient descent* or *Newton's method* can be used.

The gradient of cross entropy loss is:

$$g = \frac{d}{dw} \text{CE}(w) = \sum_i (\mu_i - y_i) x_i = X^T (\mu - y)$$



## Logistic Regression IV

where  $\mu$  should be understood as element-wise sigmoid function on  $Xw$ .

### Question

Can you derive the gradient?

Then the *gradient descent* is the following iteration:

$$w_{k+1} = w_k - \eta \cdot g_k$$

where  $\eta$  is a *hyperparameter* that needs to be determined before training (we will come back to this point later).

### Question

What happens when  $\eta$  is too large or too small?



# Outline

## Supervised Learning

Overview

Linear Regression

Logistic Regression

Support Vector Machine



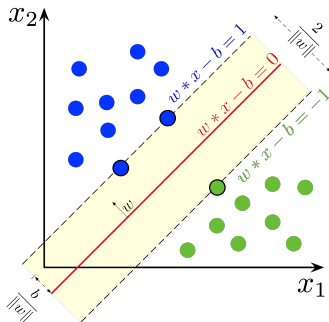
# Support Vector Machine I

Support vector machines (SVMs) are **supervised** *max-margin* models with associated learning algorithms that analyze data for **classification** and **regression** analysis.

In addition to linear classification, SVMs can perform **non-linear** classification using the kernel trick (we will not cover this in lecture).



## Support Vector Machine II



**Figure:** Max-margin idea in SVM. Source: By Larhmam - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=73710028>

We want to find a hyperplane (the red line in 2D) that divides the two groups of data (blue and green dots). E.g., the distance between the hyperplane and the nearest point (the support vector) from either group is maximized.



## Support Vector Machine III

If the training data is linearly separable (all the data can be eventually perfectly classified!), we can have two parallel hyperplanes that separate the two classes of data with largest distance. The largest distance is the margin.

When we have a normalized dataset (e.g., all data falls in range  $[0, 1]$ , etc), the two parallel hyperplanes can be written as

$$w^T x - b = 1$$

$$w^T x - b = -1$$

The distance between two hyperplanes is  $\frac{2}{\|w\|}$ . Therefore, maximizing the margin is equivalent to minimizing  $\|w\|$ . The distance requires some math to derive, feel free to explore online<sup>2</sup>.



# Support Vector Machine IV

We also need to put data points outside of the margin area, e.g.,

$$\begin{aligned}w^T x_i - b &\geq 1, & y_i &= 1 \\w^T x_i - b &\leq -1, & y_i &= -1\end{aligned}$$

which can be compactly written as

$$y_i(w^T x_i - b) \geq 1, \quad \text{for any data}$$

Now we have the objective (to maximize the margin) and constraints (to put data outside the margin area), we can formulate a **optimization problem**:

$$\begin{aligned}\min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y_i(w^T x_i - b) \geq 1, \quad \text{for any data}\end{aligned}$$



# Support Vector Machine V

## Questions

Is the SVM formulation a convex optimization problem, why?

As we mentioned earlier, there are some extensions to the hard-margin SVM:

- ▶ *Soft-margin SVM* allows some data inside the margin area.
- ▶ *Nonlinear or kernel SVM* allows to separate more complex data pattern.

In the coursework, you can test all the different settings (or hyperparameters) using Python packages such as `sklearn`<sup>3</sup>, without knowing the math behind them.

---

<sup>2</sup><https://www.akshayagrawal.com/lecture-notes/html/hyperplanes.html>

<sup>3</sup><https://scikit-learn.org/1.5/modules/svm.html> ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺



# Outline

## Unsupervised Learning

### Overview

### Principal Component Analysis

### k-means Clustering



# Unsupervised Learning: Definition I

## Questions

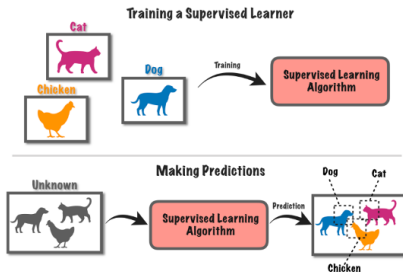
Given a set of power system measurement data, let us consider a task to separate FDI attacked measurements. What can you do if there is **no label** given?

## Definition

In *unsupervised learning*, there is only given inputs  $\mathcal{D} = \{x_i\}_{i=1}^N$  and the goal is to find interesting patterns in the data. This is a much less well-defined problem, since we are not told what kinds of pattern to look for.



## Unsupervised Learning: Definition II



**Figure:** Supervised vs unsupervised learning. Source: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-in-2-minutes-72dad148f242>.

Some examples include *principal component analysis (PCA)*, *k-means*, etc.



# Outline

## Unsupervised Learning

Overview

Principal Component Analysis

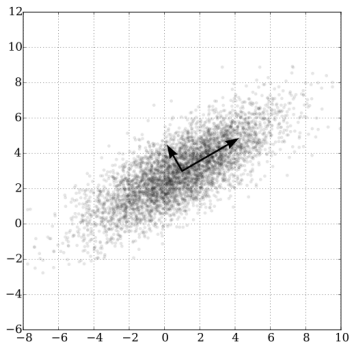
k-means Clustering



# PCA I

## Question

Consider the 2D data shown below. For the two-marked directions, which direction contains more information, why?





## PCA II

PCA<sup>4</sup> is a **linear dimensionality reduction** method that can project the data into fewer dimensions (the  $D$ ) in a way that most of the information of the data is kept!

- ▶ Minimize the dimension while keeping the information.

Consider that your power system measurement data has large number of features, e.g.,  $D$  is very large,

- ▶ You would like to visualize the data. Then...
- ▶ You would like to first extract useful information, then apply ML algorithms. Then...

Hint: You can also try PCA in coursework!

---

<sup>4</sup><https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>



# Outline

## Unsupervised Learning

Overview

Principal Component Analysis

k-means Clustering



## k-means Clustering

As the name suggests, the k-means<sup>5</sup> method divides the training set into  $k$  different classes in an **unsupervised** way.

The k-means algorithm works as follows

1. Determine the number of clusters  $k$ .
2. Initialize  $k$  different center of clusters  $\{c_1, \dots, c_k\}$ .
3. Assign each training example to cluster  $i$ , where  $i$  is the index of the nearest center  $c_i$ .
4. Each center  $c_i$  is updated to the mean of all training examples assigned to cluster  $i$ .
5. Repeat step (3)-(4).

### Question

For the task of detecting FDI attacks from unlabeled data, how many clusters should be assumed in step (1)?

---

<sup>5</sup>[https:](https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html)

[//scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html](https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html)



# Outline

## Reinforcement Learning Overview Only



# Reinforcement Learning

There is a third type of machine learning, known as *reinforcement learning*, which is somewhat less commonly used. This is useful for learning how to act or behave when given occasional reward or punishment signals. We will not cover this type of ML in this lecture and the coursework.



# Outline

## Performance Measure

### Overview



## Performance Measure I

In order to evaluate the abilities of a machine learning algorithm, we must design a quantitative but straightforward measure of its performance.

For *regression task*, we can use the mean squared error (MSE) as defined before.

For the binary classification task on FDI attack, we can define more performance measures, including

- ▶ *True positive rate (TPR)*: the proportion of FDI attacks (class 1) that are correctly marked (as class 1). You can also call it the detection rate.
- ▶ *False positive rate (FPR)*: the proportion of normal measurement (class 0) that are incorrectly marked (as class 1). You can also call it false alarm.
- ▶ *True negative rate (TNR)*: the proportion of normal measurements (class 0) that are correctly marked (as class 0).



## Performance Measure II

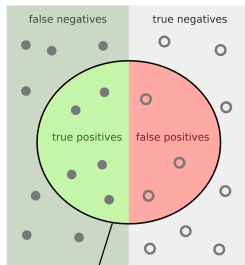
- ▶ *False negative rate (FNR)*: the proportion of FDI attacks (class 1) that are incorrectly marked as (class 0). You can also call it missing alarm rate.

To systematically include all the measures, the  $F_1$  score is defined as

$$F_1 = \frac{2TP}{2TP + FP + FN}$$



## Performance Measure III



**Figure:** Performance measure for classifications. The left part represents positive samples and the right part represents the negative samples. The cycle represents the detected positive samples.

### Question

Can you identify the TPR, FPR, TNR, and FNR in the above plot?



# Outline

Model Capacity

Underfit and Overfit

Regularization



# Overfitting and Underfitting I

Having good accuracy on the training dataset is not the ultimate goal of machine learning. The central challenge in machine learning is that we must perform well on **new, unseen** inputs. The ability to perform well on previously unobserved inputs (you can call it test set) is called *generalization*.

When training a machine learning model, we **only** have access to a training set, we can compute some error measure on the training set called the *training error*, and we reduce this training error as described in the linear regression case through optimization.

$$\text{MSE}^{\text{train}} = \|X^{\text{train}}w - y^{\text{train}}\|_2^2$$

What separates machine learning from optimization is that we want the generalization error, also called the *test error*, to be low as well.

$$\text{MSE}^{\text{test}} = \|X^{\text{test}}w - y^{\text{test}}\|_2^2$$



# Overfitting and Underfitting II

Therefore, the factors determining how well a machine learning algorithm will perform are its ability to:

1. Make the training error small.
2. Make the gap between training and test error small.

## Definition

*Underfit* occurs when the model is not able to obtain a sufficiently low error value on the training set. *Overfit* occurs when the gap between the training error and test error is too large.



## Overfitting and Underfitting III

We can control whether a model is more likely to overfit or underfit by altering its *capacity*. Models with insufficient capacity are unable to solve complex tasks. Models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task, they may overfit. One way to control the capacity of a learning algorithm is by choosing its *hypothesis space*, the set of functions that the learning algorithm is allowed to select as being the solution.

As mentioned before, the linear regression algorithm has the set of all linear functions of its input as its hypothesis space. We can generalize linear regression to include polynomials, rather than just linear functions, in its hypothesis space. Doing so increases the capacity of the model.



## Overfitting and Underfitting IV

Normally, we can start to choose model with **lower** capacity (complexity).  
For example, a linear regression with polynomial of degree one:

$$\hat{y} = b + wx$$

By introducing  $x^2$  as another feature, we increase the complexity of the regression model:

$$\hat{y} = b + w_1x + w_2x^2$$

We can continue to add more powers of  $x$ , for example a polynomial of degree of 9:

$$\hat{y} = b + \sum_{i=1}^9 w_i x^i$$



# Overfitting and Underfitting V

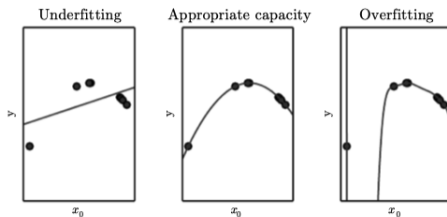


Figure: Underfit vs overfit



## Overfitting and Underfitting VI

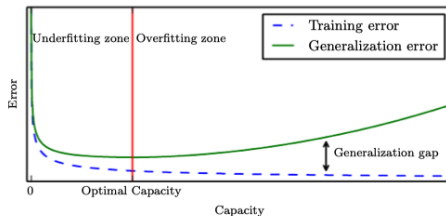


Figure: Typical relationship between capacity and error.

### Question

We still have many unsolved problems. For example, we don't have access to the test set... Then how can we know the generalization error (or the performance on test dataset) during training?



# Outline

## Model Capacity

Underfit and Overfit

Regularization



## Regularization I

**Degree of complexity:** So far, the only method of modifying a learning algorithm is to increase or decrease the model's representational capacity by adding or removing the degree of the features.

**Larger hypothesis space:** Another way to control the performance (capacity) of the algorithms is to choose what kind of functions that can better fit on the data. For example, linear regression would not perform very well if we try to use it to fit on  $\sin(x)$  from  $x$ .

**Regularization:** We can also give a learning algorithm a preference for choosing one solution in its hypothesis space to another. I.e., the unpreferred solution will be chosen only if it fits the training data significantly better than the preferred solution.

We can include the *weight decay* on the linear regression loss for *regularization* purpose.

$$\text{MSE}^{\text{reg}} = \text{MSE}^{\text{train}} + \lambda w^T w$$



## Regularization II

where  $\lambda$  is a value chosen before training (a hyperparameter).

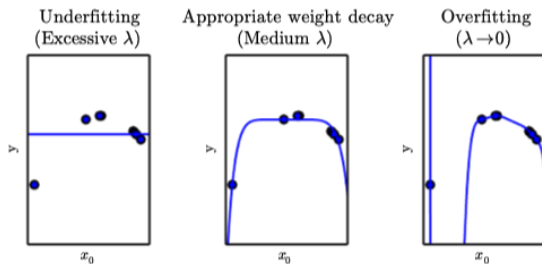


Figure: Different  $\lambda$  on the overfitted polynomial.



# Outline

## Hyperparameters and Validation Set

### Hyperparameters

### Validation Set



# Hyperparameters I

## Definition

Most machine learning algorithms have several settings that we can use to control the behavior of the learning algorithm. These settings are called hyperparameters. The values of hyperparameters are **not** adapted by the learning algorithm itself.

The hyperparameters significantly influence the performance of the ML model. For example,

1. In the polynomial regression,  $\phi(x) = [1, x, x^2, \dots, x^d]$ , the degree of the polynomial, which represents the **capacity** hyperparameter.
2. In the regression with weight decay,  $\text{MSE}^{\text{reg}} = \text{MSE}^{\text{train}} + \lambda w^T w$ , the  $\lambda$  value used to control the strength of weight decay is another example of hyperparameter.
3. Whenever you train an ML model, there always exists a set of hyperparameters which should carefully play with.



# Hyperparameters II

## SVC

```
class sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale',  
coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200,  
class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr',  
break_ties=False, random_state=None) \[source\]
```

**Figure:** Sklearn SVM function. There are many hyperparameters to be set.

## Question

In the simplest setting, it is not appropriate to learn the hyperparameters on the training set. Based on our discussion on overfitting, explain why.



# Outline

## Hyperparameters and Validation Set

Hyperparameters

Validation Set



# Validation Set I

Recall a previous problem,

## Question

We still have many unsolved problems. For example, we don't have access to the test set... Then how can we know the generalization error (or the performance on test dataset) during training?

We would like to **estimate** the generalizability of the ML model in **an unknown dataset**. The solution is to separate a held-out set from the **training dataset**, called *validation set*. Important notice:

1. The validation set should not be used to train the model!
2. No test data should be included in the validation set!

Specifically, we split the training data into two disjoint subsets. One of these subsets is used to learn the parameters (80%). The other subset is our validation set (20%), used to estimate the generalization error during or after training, allowing for the hyperparameters to be updated accordingly.



## Validation Set II

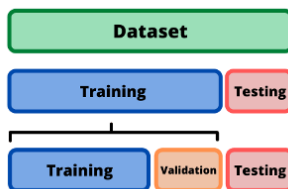


Figure: Train, validation, and test dataset.

For example, in the coursework, you will be asked to train an ML detector to detect FDI attacks. You will be provided with labeled train dataset and unlabeled test dataset.

1. Split the training dataset into 'training dataset' and validation set with a predefined ratio (0.8 vs 0.2).
2. Determine an ML model you would like to train on.

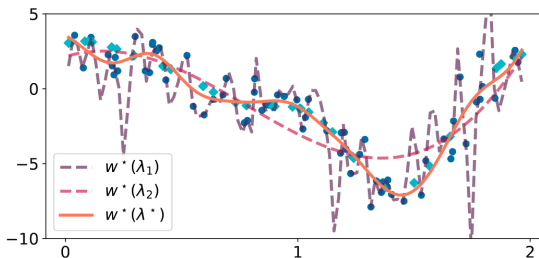


## Validation Set III

3. Set a list of value of all hyperparameters. E.g.,  $\lambda = [0, 0.01, 0.001]$ ,  $d = 1, 2, \dots$  etc.
4. Train the ML model under each of the hyperparameters combination, e.g.  $(\lambda = 0, d = 1)$ ,  $(\lambda = 0.01, d = 2)$ , etc on the **training dataset**. This processing is also called *grid search*.
5. Evaluate the performance of the 9 trained models on **validation set**. The one with the best performance is the final model.



## Validation Set IV



**Figure:** Tuning hyperparameter  $\lambda$  based on the performance on the **validation set**. This example is taken from [Blondel and Roulet, 2024]. Blue dots are training samples and cyan dots are validation samples.

### Question

Can you tell which fitting curve is over-fitted and under-fitted?



# Reference I



Blondel, M. and Roulet, V. (2024).

The elements of differentiable programming.

*arXiv preprint arXiv:2403.14606.*



Hamann, H. F., Brunschwiler, T., Gjorgiev, B., Martins, L. S., Puech, A., Varbella, A., Weiss, J., Bernabe-Moreno, J., Massé, A. B., Choi, S., et al. (2024).

A perspective on foundation models for the electric power grid.

*arXiv preprint arXiv:2407.09434.*