Week 1: Introduction

Yunwen Lei

School of Computer Science, University of Birmingham

1 Definition of Machine Learning

Definition 1. A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

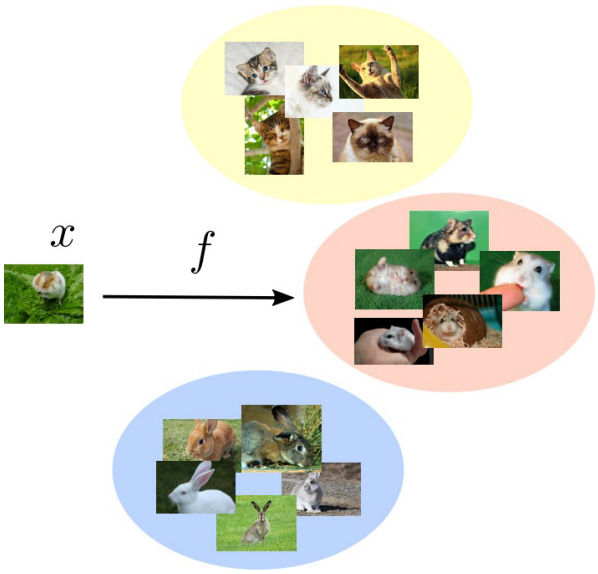
Now let us consider an example where a little girl wants to build a tower with toy blocks. The girl is very young. But she still has some experience by looking at the physical world on how a tower looks. For example, she knows she needs to put big blocks in the bottom and small blocks in the top. This experience increases as she plays the gamer more and more. The task is to build a tower as high as possible. An appropriate performance measure is the height of this tower. Therefore, in this example we have all the three components of a machine learning problem as follows

- E: physical world
- T: building a tower with toy block
- P: how tall the tower is

1.1 Machine Learning Tasks T

We have several machine learning tasks. Below we mention some common learning tasks.

Classification. For this learning task, the experience is given in terms of training examples. Each training example involves an input-output pair. The input $\mathbf{x}$ gives some feature representation, e.g., the color, the shape, the weight and so on. The output y takes values in a prescribed set $\{1, 2, ..., k\}$, where k is the number of classes. Each number can have its own meaning (e.g., we can use 0 to mean cat, 1 to mean dog, 2 to mean a bird and so on). If $k = 2$ this becomes a binary classification problem. Otherwise, we have a multi-class classification problem.



- Construct a function

$$f : \mathbb{R}^d \mapsto \{1, \ldots, k\}$$

such that if an object with features $\mathbf{x} \in \mathbb{R}^d$ belongs to class y then

$$f(\mathbf{x}) = y$$

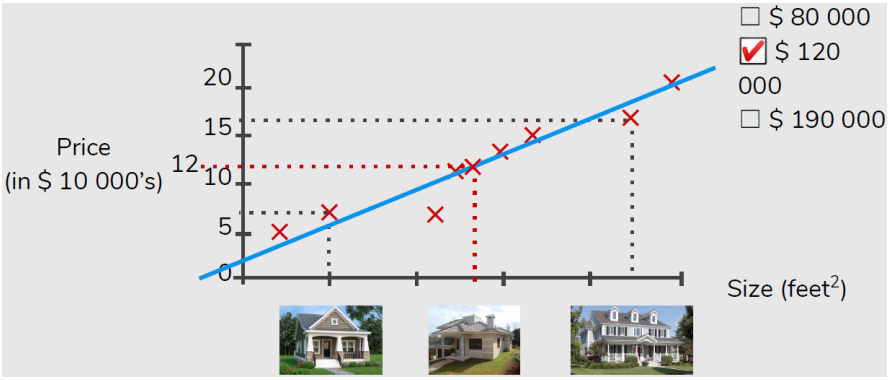
- Alternatively, construct a function which given features returns the probability of each class

Regression. For this learning task, the experience is given in terms of training examples. Each training example involves an input-output pair. The difference is that the output can be any real-number. Our aim is to predict a numerical output given some input, i.e., a function

$$f : \mathbb{R}^d \mapsto \mathbb{R}.$$

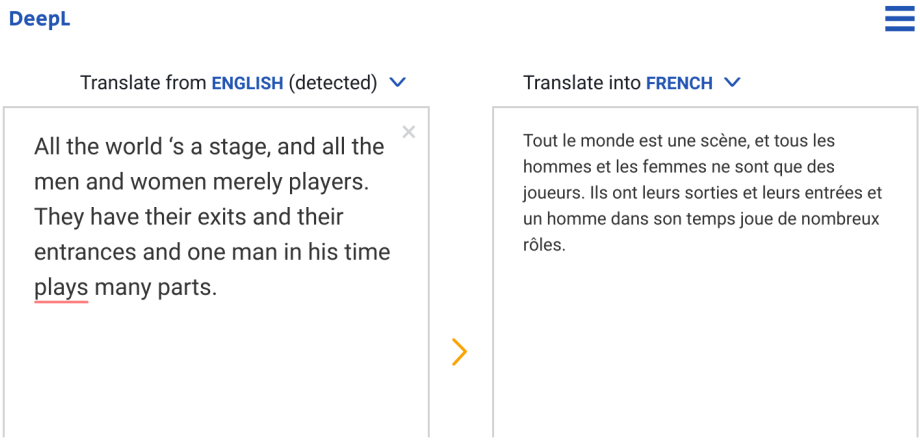
Example 1 (House price prediction)**.** For house price prediction problem, we aim to predict the price of a house according to some feature information

- **Input**: Information of House (living size, lot size, location, # floors)
- **Output**: Price

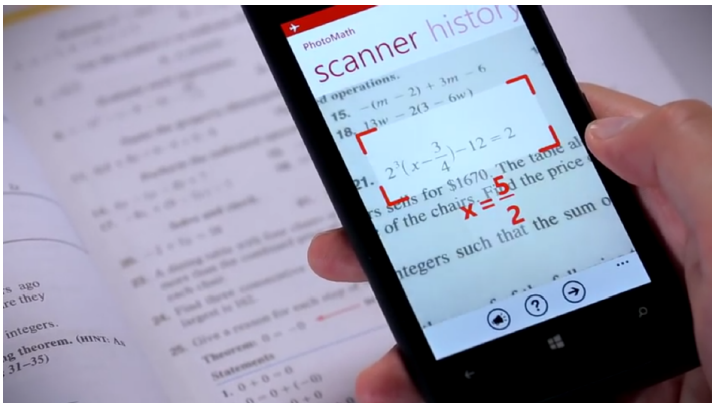


Machine Translation. For this task, we aim to translate from a source language to a target language. This is a learning task in the domain of *natural language processing*, and has found wide applications. This is not easy since difference language have different grammer and is not structured. Unlike classification and regression, the output in this task is complex.

- **Input**: sequence of characters (e.g., English text)
- **Output**: sequence of characters (e.g., French text)



Photomath. Photomath is a mobile computer algebra system with an augmented optical character recognition system designed for use with a smartphone's camera to scan and recognize mathematical equations; the app then displays step-by-step explanations onscreen.



1.2 Experience E and Performance P

Experience E: we get experience by observing a dataset from nature. As we see before, experience is often expressed in terms of training datasets. Depending on the form of dataset, we have unsupervised learning (only input features) and supervised learning (input-output pairs).

- unsupervised learning:  ,  ,  , ...

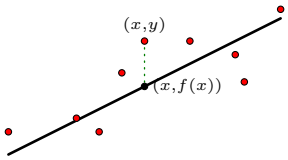
  or formally $S = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\}$

- supervised learning: ( ,`dog`), ( ,`car`), ( ,`airplane`), ...

  or formally $S = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$

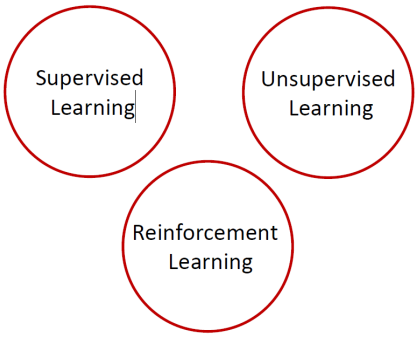
Performance P: once we find a solution for the learning task, we need a performance measure to quantify the performance of the solution.

- **Accuracy** is a common performance measure for classification tasks
  - Proportion of correctly classified examples, i.e., the number of correctly classified examples divided by the total number of examples
- **Residual** is a common performance measure for regression tasks, which is the difference between the true output y and the predicted output $f(x)$ for a model f on an example (x, y)



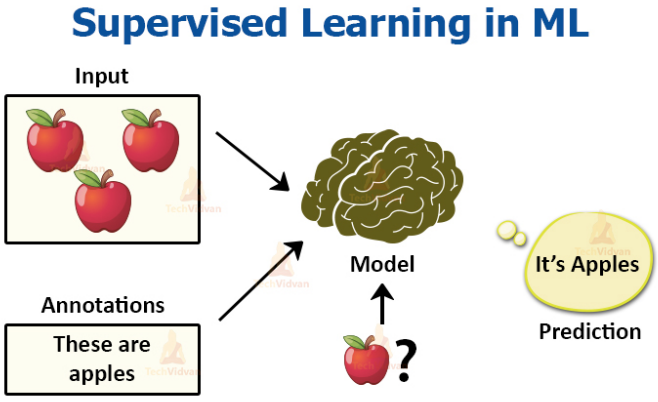
1.3 Types of Learning Tasks

- **Supervised Learning**: we have a training dataset in terms of input-output pairs, the output is the *supervised* information. We aim to build a model mapping an input to an output so that the predicted output is similar to the true output.
- **Unsupervised learning**: we only have feature information and no supervised information. We aim to find a pattern among the feature information (e.g., partition the dataset into different groups so that examples within the same group are similar and examples in different groups are not similar) or an internal representation of the input (we can transform a high-dimensional data to a low-dimension so that the relationship between examples are preserved under this transformation)
- **Reinforcement learning**: this is a learning task where we want to interact with the environment. Our goal is to maximize some reward function.



Definition 2 (Supervised Learning)**.** Learn a function that maps an input to an output based on examples of input-output pairs (the supervised information is the output).

In the following example, the input information are images and the output information are annotations. Once given several input-output pairs, we aim to build a model. Then we can use it to do prediction: once we are given a new image, we can predict whether it is an apple.



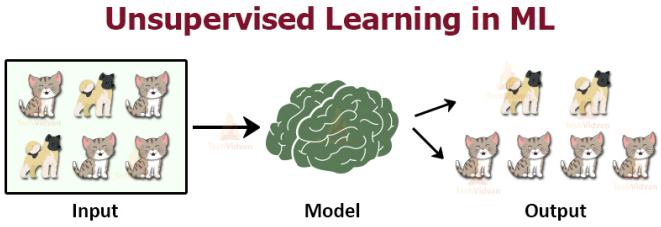
Two typical tasks in supervised learning are classification and regression

- Classification is used to predict discrete values (class labels).

- Regression is used to predict continuous values

Definition 3 (Unsupervised Learning)**.** Learn interesting patterns from dataset with no labels: $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ (we have no supervised information).

In the following example, we have several inputs as images. We aim to build a model to partition the dataset into two groups: one for cats and one for dogs. There is no supervised information in this learning process. The algorithm can automatically use the similarity information to do this partition.



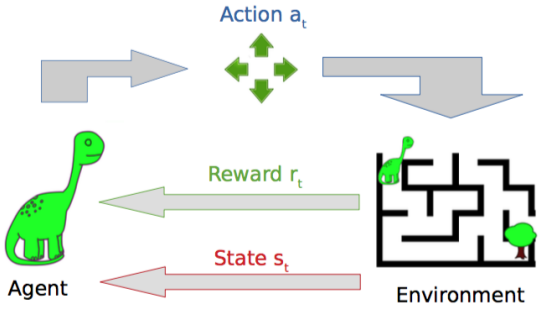
Two typical tasks in supervised learning are clustering and dimensionality reduction

- **Clustering** algorithm tries to detect similar groups. In this task, we are given a similarity measure to quantify the similarity between different examples. We aim to find a partition of the training dataset into different groups (clusters): examples within the same group are similar under this similarity measure, examples from different groups are dissimilar.
- **Dimensionality reduction** tries to simplify the data without losing too much information. Often an example has many features. Some feature are important for learning, while some features are not. Dimensionality reduction aims to transform the original features to some few features, while still keeping as much information as possible.

Definition 4 (Reinforcement learning)**.**

- An agent interacts with an environment (e.g., game of maze)
- In each time step
  - the agent receives **observations** (e.g. (x, y)) which give it information about the **state** (e.g. positions of the dinosaur)
  - the agent picks an **action** (e.g. moving direction) which affects the **state**
- The agent periodically receives a **reward** (e.g, -1 per step)
- The agent wants to learn a **policy**, or mapping from observations to actions, which maximizes its average reward over time

Example 2. Let us consider an example of escaping the maze. The agent observes her location (x, y) which is a state. The agent can take some action, i.e., one step to the north, south, east or west. After taking this action, the state would change. For example, if she chooses to go the north, the state will be $(x, y+1)$. Meanwhile, the environment will give she a feedback in terms of reward. In this example, we can choose the reward as -1 per step since we want to escape the maze as quickly as possible. The agent chooses her action according to a policy. The policy is a map from the state to actions. The agent needs to learn the policy so that she can achieves the maximum rewards over time.



2 Training and Testing

2.1 Training and Testing

Some important concepts in ML include training and testing. We first explain these concepts by the example of house price prediction.

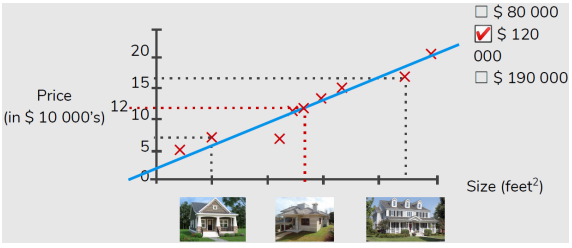
- **Input**: Information of House (living size, lot size, location, # floors)
- **Output**: Price
- **Aim**: find a model to predict price based on feature information of house
- **Training dataset**: a sequence of (features, price) pairs, which is used to build a model mapping house information to price

$ 70,000

$ 160,000

- **Prediction/testing**: given information of new house, we use the built model to predict its price

???



- **Performance**: difference between predicted price and true price (we wish the difference to be as small as possible)

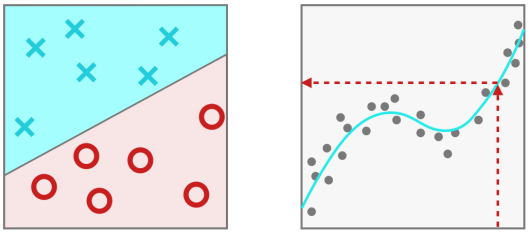
Now we give a more formal introduction on the training and testing.

- **Training dataset**: a dataset that contains n samples.

$$(\underbrace{\mathbf{x}^{(1)}}_{\text{input}}, \underbrace{y^{(1)}}_{\text{output}}), (\underbrace{\mathbf{x}^{(2)}}_{\text{input}}, \underbrace{y^{(2)}}_{\text{output}}), \ldots, (\underbrace{\mathbf{x}^{(n)}}_{\text{input}}, \underbrace{y^{(n)}}_{\text{output}})$$

The input give some features while the output can be a label (classification) or a real number (regression)

- Classification: $y \in \{-1, +1\}$
  $+1$ means positive examples
  -1 means negative examples
- Regression: $y \in \mathbb{R}$



- Aim to find a function $f : \mathcal{X} \mapsto \mathcal{Y}$ such that

$$y \approx f(\mathbf{x})$$

- **Prediction**: given a new input $\mathbf{x}$, use f to do prediction
  - if a house has x square feet, predicting its price?

2.2 Loss function

Another very important concept in ML is loss function. Once we build a model we are interested in its performance on the prediction. A loss function provides a way to quantify the performance of a model on a single training example. Generally, a loss function scores how far off a prediction is from the desired "target" output. The loss function takes two arguments: the first is the predicted value and the second is the target (true) output. Then Loss(prediction,target) returns a number called "the loss". Big loss function means a bad performance of the model on the example. Small loss means a good performance of the model on the example. A zero loss means there is no error by using the model to do prediction on the example.

- If we consider classification, we can use the 0-1 loss as follows

$$\text{Loss}(\hat{y}, y) = \begin{cases} 0, & \text{if } y = \hat{y} \\ 1, & \text{otherwise.} \end{cases},$$

  where y is the true output label and $\hat{y}$ is the label predicted by the model on the example (x, y). Then $\text{Loss}(\hat{y}, y) = 1$ means the prediction is wrong, while $\text{Loss}(\hat{y}, y) = 0$ means the prediction is correct.

- If we consider regression, we can use the square loss as follows

$$\text{Loss}(\hat{y}, y) = (y - \hat{y})^2.$$

2.3 Evaluating a Prediction Function

Suppose we have a data science intern who trains a model on a dataset and gives you a prediction function $f(x)$. This model achieves an on-average error on training data of less than 1%. This shows the good behavior of the model on training data. The product manager says that "we can deploy the model if we can achieve error less than 2%". There comes a question

Can we deploy this prediction function?

At first glance, you may think we can deploy this model since its error 1% is smaller than the requirement. However, this does not give a convincing argument: the error 1% is achieved for the training dataset. What we really want is that it has a good performance on new inputs (testing). The behavior on training can be very different from the behavior on testing. Indeed, since we use the training dataset to train our model, the model has a bias towards on the training dataset. In this case, the training set is no longer appropriate to measure the performance of the model.

The only way to know how well a model will generalize to new cases is to actually try it out on new cases

- Training set: only for **training** prediction functions
- Test set: only for **assessing performance** (**independent** of training)

Remark 1. Performance on training set has a bias towards the model and cannot serve as an unbiased estimate of its performance on true environment!

This leads to two important concepts: the training error and test error.

Definition 5 (Training error)**. Training error** is the error for using a model f to do prediction on **training data**

$$\text{Err}_{train}(f) = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}(f(\mathbf{x}^{(i)}), y^{(i)})$$

For house price prediction, this can be

$$\text{Loss}(f(\mathbf{x}^{(i)}), y^{(i)}) = (f(\mathbf{x}^{(i)}) - y^{(i)})^2.$$

Definition 6 (Testing error)**. Testing error**: the error for using a model f to do prediction on **test data**. Suppose we have a testing dataset $(\tilde{\mathbf{x}}^{(1)}, \tilde{y}^{(1)}), \ldots, (\tilde{\mathbf{x}}^{(m)}, \tilde{y}^{(m)})$. The testing error becomes

$$\text{Err}_{test}(f) = \frac{1}{m}\sum_{i=1}^{m} \text{Loss}(f(\tilde{\mathbf{x}}^{(i)}), \tilde{y}^{(i)})$$

We are mostly interested in the test error, which can be decomposed as follows

$$\text{Err}_{test}(f) = \text{Err}_{train}(f) + \underbrace{\text{Err}_{test}(f) - \text{Err}_{train}(f)}_{\text{Gen}_{\text{gap}}(f)}$$

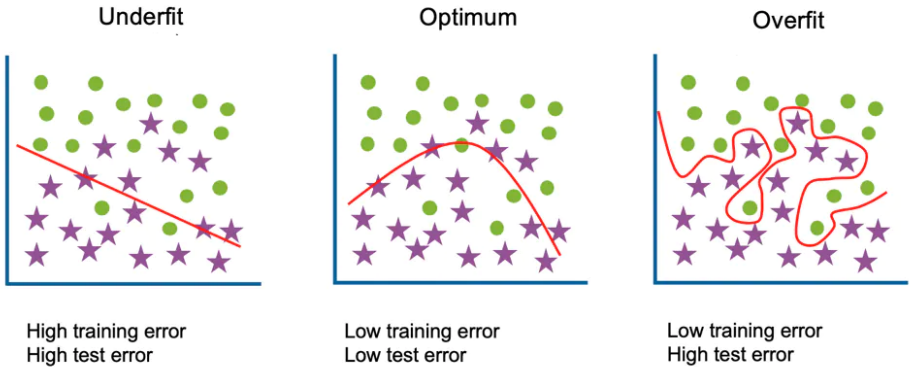
- Typically, the generalization gap $\text{Gen}_{\text{gap}}(\text{f})$ is greater than 0. The reason is that the model is trained on the training dataset, and therefore it can achieve very small training error.
- Typically, a simple model has a small $\text{Gen}_{\text{gap}}(f)$. A simple model means the model is not that biased.
- To achieve a small test error, we require a small training error and meanwhile a small generalization gap. That is, we want the model to have a good performance on training and meanwhile not complex.

3 Underfitting and Overfitting

The different behavior on training and testing leads to concepts of underfitting and overfitting.

Definition 7.
- Loosely speaking, we say a model underfits when its training performance is poor.
- We say a model overfits when its training performance is good but test performance is poor.

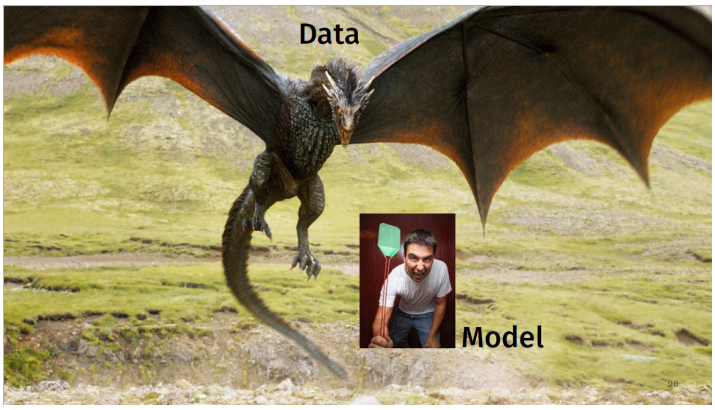
Below we give some examples to explain underfiting and overfitting. We consider a classification problem. In the left figure, we consider a linear model. This model is too simple to have a good performance on the training examples. In the right figure, we consider a very complex model. This model is very clever in achieving no mistakes in the training dataset. However, this model is so complex that it is not clear whether this behavior would extend to new examples. We would expect that the true model should not be as complex as this. This leads to overfitting. In the middle we find a model we want. Although it does some mistakes in training, its training performance is still good. Meanwhile its complexity is not high and we expect the behavior on training can generalize to new examples.



Remark 2 (Overfitting the training data)**.** Now we give an example to explain overfitting. Suppose we want to build a bed. On the one hand we wish this bed is convenient to sleep. On the other hand, we wish the bed is cheap (using as little material as possible). If we only use the sleeping habit of a single person and we want very good behavior on this person, we may build a bed as shown in this figure. As you can see, this bed is perfect for this person because this person sleeps in this shape. Furthermore, this bed is very cheap. However, this bed overfitts this person. If another person wants to sleep, she would find the bed useless.

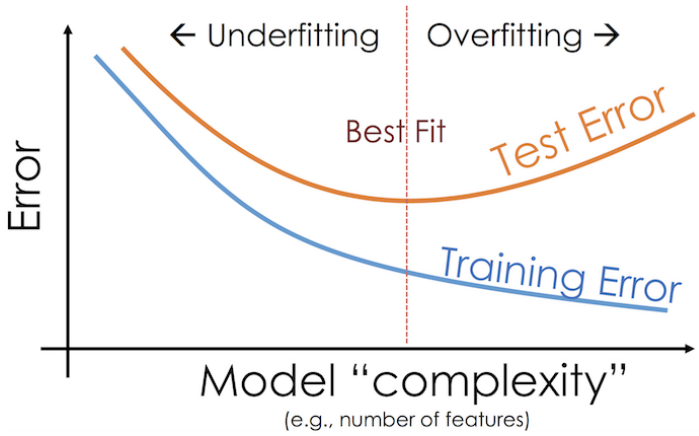
- Overfitting means that the model performs well on the training data but it does not generalize to testing data
- It happens when the model is too complex relative to the amount of the training data.

Remark 3 (Underfitting the training data)**.** Underfitting is the opposite of overfitting: it occurs when your model is too simple to learn the underlying structure of the data. We explain this phenomenon through the following figure. In this example, the data is very complex (something similar to a huge animal) while the model is too naive (something like a fly swatter). Of course, this fly swatter is not enough to beat the huge animal



Remark 4 (Underfitting and Overfitting)**.**

- In general, the training error decreases as we add complexity to our model with additional features or more complex prediction mechanisms.
- The test error, on the other hand, decreases up to a certain amount of complexity then increases again as the model overfits the training set.
- We need to balance the behavior on training dataset and the simplicity to get a model with best fit (i.e., finds a suitable model complexity: complex enough to have a good behavior on training and not too complex for a good generalization to testing)

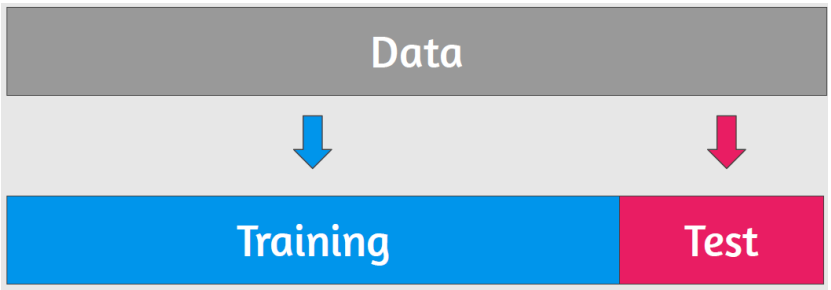


4 Basic Machine Learning Workflow

4.1 Data Partition

It is common to use 80% of the data for training and hold out 20% for testing

- Training set is used only for **training** prediction functions
- Test set is used only for **assessing performance** (**independent** of training)



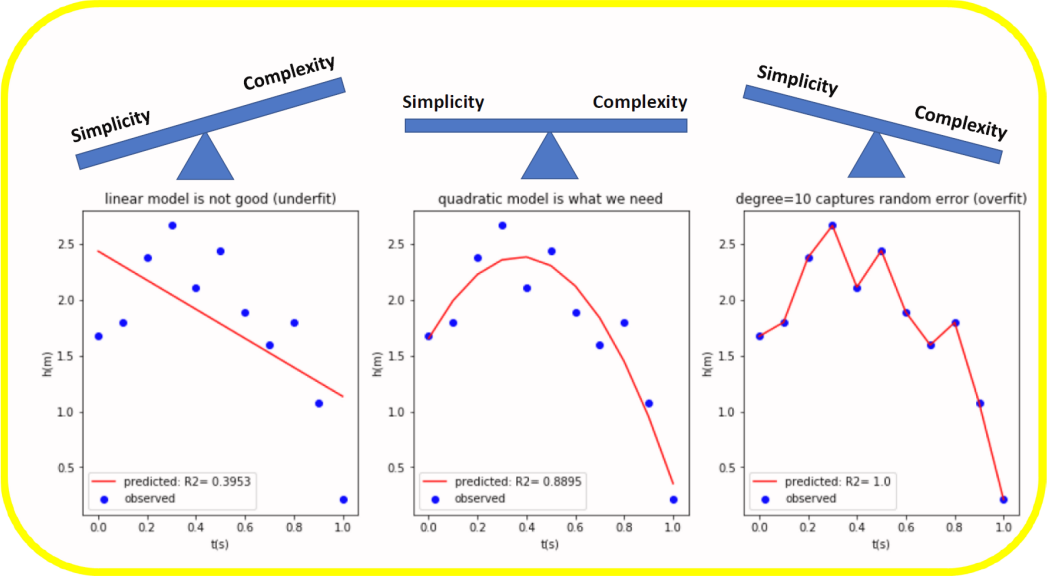
Suppose you want to train a model for deployment. Here is what you can do to avoid underfitting/overfitting.

- Give the training set to data science intern, you keep the test set
- Intern will train a model on the training set and get a model. She then gives a prediction function (model) to you.
- You evaluate prediction function on test set
- No matter what intern did with training set, test performance should give you good estimate of deployment performance. This is because the test set does not participate in the learning process.

4.2 How to Find a Good Model?

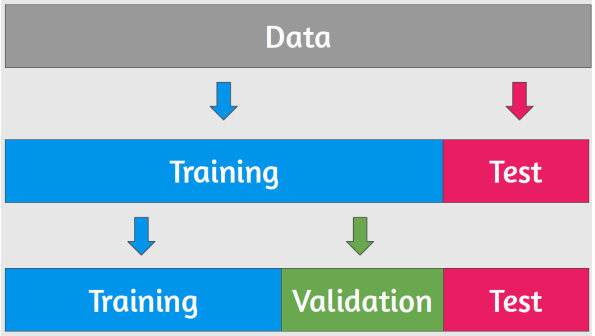
*Intern wants to try **many** fancy ML models: each gives a different prediction function, e.g., she can train a linear model or a neural network. Even with the same method, she can get different models with different parameters, e.g., neural networks with three layers and four layers. Among these models, which model she should choose for deployment?*

We need to find an appropriate level of complexity: if the model is too simple, then it has bad behavior even on the training dataset. If it is too complex, it is likely to have a good behavior on training but bad behavior on testing. In the right figure we show that we can interpolate the training dataset by choosing a very complex model. However, this model is not idea for prediction. In the middle is the model we want: we want to find a balance between simplicity and complexity.



This observation motivates a further partition of the dataset. Instead of training and testing, we preserve some portion of dataset for validation. Below is a typical process.

- Intern needs her own test set to evaluate prediction functions.
- Intern should randomly split data again into (80/20 split): 80 percents for training and 20 percents for validation
- Validation set is like test set: it is also used for valuating the performance of the model. Unlike testing dataset, it is used to choose the best model among many prediction functions.
- Test set is just used to evaluate the final chosen prediction function.



Example 3. Suppose we train several neural networks with different layers. The basic process of using a validation data set to choose a model with a good architecture is:

Since our goal is to find the network having the best performance on new data, the simplest approach to the comparison of different networks is to evaluate the error function using data which is independent of that used for training. Various networks are trained by minimization of an appropriate error function defined with respect to a training data set. The performance of the networks is then compared by evaluating the error function using an independent validation set, and the network having the smallest validation error is selected. This approach is called the hold out method. Since this procedure can itself lead to some overfitting to the validation set, the performance of the selected network should be confirmed by measuring its performance on a third independent set of data called a test set.

4.3 Basic Machine Learning Workflow

Here is a basic machine learning workflow

1. Split labeled data into **training**, **validation**, and **test** sets
2. Repeat until happy with performance on validation set
   - Choose some ML algorithm
   - Train several ML models with various complexities
   - Evaluate prediction functions on validation set and choose the best one
3. Evaluate performance on test set