Week1 Introduction to Machine Learning

- **Machine learning process**: Learn a model of functional dependence between input attributes and target values from the training set, and use it to predict target values of unknown(unseen) data
    - Need to **determine the parameters of a model** (parametric model)
    - Need to **well fit the model** to the training by minimizing an error function between predicted-target values and correct-target values
    - Need to **prevent overfitting**:
        - the order of complexity M, the regularisation parameter λ
        - Using **cross-validation** (K-fold)
        - **RMS** for both training error and testing error

> The distinguishing point in learning between human and machine: Human learn from **experience**, machine learn patterns from large volume of **data**.

- 2 main **purpose** of machine learning:
    - **Prediction**: predict the target/response value for any (new) value of the attribute variables
    - **Inference**: Understand the way **how the target variable is affected as the input variables change**
- Type of relationship: **linear** relationship v.s **Non-linear** relationship
    - **Linear relationship**: Only **one parameter** associated with input x

      $t = w_0 + w_1 x$
    - **Non-linear relationship**: Input x associated with **mutiple parameters**

      $t = w_0 + w_1 x + \cdots + w_M x^M = w_0 + \sum_{j=1}^{M} w_j x^j = \sum_{j=0}^{M} w_j x^j$
    - Learning involves in using the data to choose suitable values for parameters w, and validate the setting:
        - **Training set**: used to learn the parameters
        - **Testing set**: used to test whether the trained model correctly predict the target variables.
    - **Generalization**: The ability in predicting the target value **for new examples** (data) that differ from those used for the training. In other words, the models can **generalise well to unseen examples**.
- **Supervised learning v.s Unsupervised learning**
    - **Supervised learning**: In training phase, the value of the target variables of corresponding input variables are given.
    - **Unsupervised learning**: In the training phase, only the input variables are given
- **Regression v.s Classification**:
    - **Regression**: The target variables are real-valued and continuous

- **Classification**: The target variables are a finite number of discrete categories

---

Fundamental Concepts

1. Regression problem

> Remember: **Noise** inevitably exist in data.

Objectives:

- Use the training set to build (train) a model
- Discover the underlying function (The pattern)
- Assess the generalisation of the trained model by comparing the predicted value and true value of the target variables in the testing set.

Challenges:

- Need to generalise the model from a **finite data set**
- The training data are corrupted with **noise**: uncertainty exists

Model types:

- **Parametric** model: parameters are fixed regardless of the size of the training set
- **Non-parametric** model: the number of parameters can grow as the size of training set increases.

2. Polynomial Curve Fitting

- **M-polynomial function**: The general form

  $y(x, \boldsymbol{w}) = w_0 + w_1 x + \cdots + w_M x^M = w_0 + \sum_{j=1}^{M} w_j x^j = \sum_{j=0}^{M} w_j x^j$

  - $\boldsymbol{w}$ is the column vector colectively denoting the model parameters (polynomial coefficients)
  - M is the order of the polynomial

  Note that $y(x, \boldsymbol{w})$ is a **non-linear function of the input** x , but is a **linear function of the parameters** $\boldsymbol{w}$.

- **Determine** w

  The value of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimising an **error function** that **meansures the misfit** between the function $y(x, \boldsymbol{w})$ for any given value of $\boldsymbol{w}$, and the training set data points.

  ## SSE: The sum of the squares

  One simple choice of error function is given by **the sum of the squares of the errors** (**SSE**) measuring the error between the predictions for each data point x_n and the target value t_n :

  $E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x, \boldsymbol{w}) - t_n\}^2$ (*factor* $\frac{1}{2}$ *is for later convenience*)

- $y(x, \boldsymbol{w})$ is our trained model predictions
- t_n is the true target values

Hence, our training objective is to **find $\boldsymbol{w}$ that minimise the error function $E(\boldsymbol{w})$**.

$\boldsymbol{w}^* = \arg Min_{\boldsymbol{w}} E(\boldsymbol{w})$

> Since the error function is a **quadratic function** of the coefficients $\boldsymbol{w}$, its derivatives ($\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$) with respect to the coefficients will be linear in the elements of $\boldsymbol{w}$. So the minimization of the error function can be found at the stationary point in **closed form** by setting the derivatives equals to zero: $\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$

- **Model complexity**

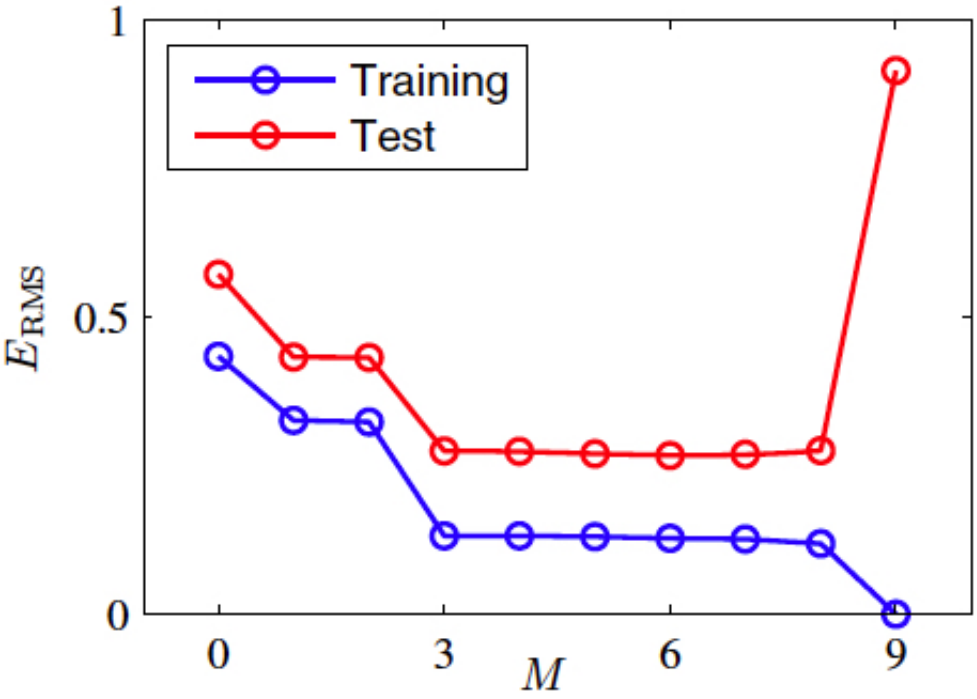
  The **order** of the polynomial function M determines the **complexity** of the model: **the higher the order, the more complex the model**.

- **Underfitting v.s Overfitting**
  - **Underfitting**: the model is too **simple** (rigid) to capture the underlying function.
  - **Overfitting**: the model is too **complex** (flexible) to capture the underlying function.

RMS: The root-mean-square

- Without knowing the underlying function, we can evaluate $E(\boldsymbol{w}^*)$ for both training and testing set to measure **the generalisation performance** by **the root-mean-square** (**RMS**) error: $E_{RMS} = \sqrt{\frac{2E(\boldsymbol{w}^*)}{N}}$

  It allows us to **compare different sizes of data sets** on an equal footing, and the square root ensures that E_{RMS} is measured **on the same scale** as the target variable t.



	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

> **As M increases, the parameters become larger**. The models then become too flexible and the parameters were very **finely tuned to the training points**. **It not only modelling the training points, but also integrate the noise into the model**.

Deal with over-fitting

For a given model complexity, there are two ways to deal with over-fitting problem:

- **Increase the size of a training set**: The over-fitting problem become less severe as the size of the data set increases, the larger the data set, the more complex the model that we can afford to fit to the data.

  > Note that the number of parameters is not necessarily the most appropriate measure of model complexity. It is more reasonable to choose the complexity of the model according to the complexity of the problem being solved.
  >
  > By least square approach in finding the model parameters, the over-fitting problem can be understood as a general property of maximum likelihood.
  >
  > By Bayesian approach, the over-fitting problem can be avoid. The model effective number of parameters adapts automatically to the size of the data set

- **Regularisation**: Adding a penalty term to the error function in order to discourage the coefficients from reaching large values. The simplest form of such penalty term is **a sum of square of all the coefficients**:

  $$E(\tilde{\boldsymbol{w}}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x,\boldsymbol{w}) - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

  - $\|\boldsymbol{w}\|^2 \equiv \boldsymbol{w}^T\boldsymbol{w} = w_0^2 + w_1^2 + w_2^2 + \cdots + w_M^2$
  - λ is the regularisation parameter governing **the relative importance** of the penalty term

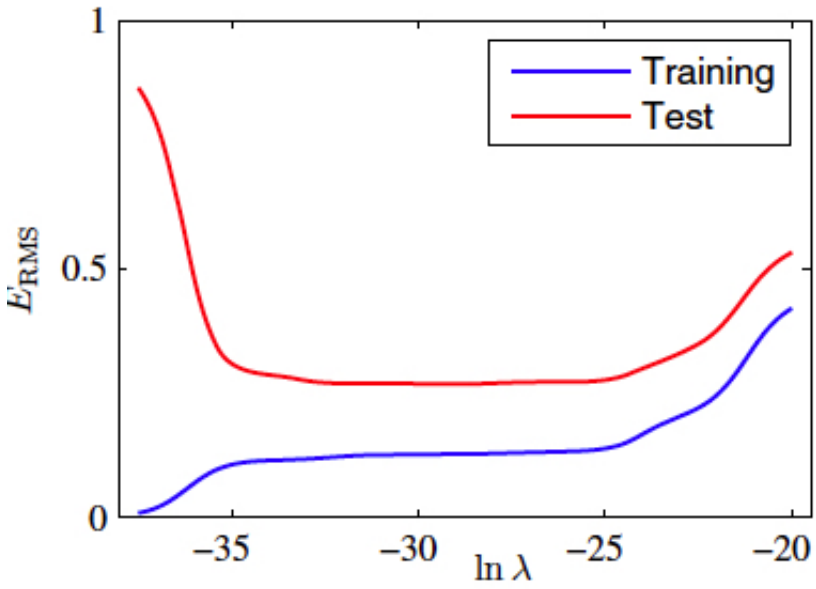
    > **The larger λ , the more important is the penalty**.

  - Regularization has the desired effect of **reducing the magnitude of the coefficients**.

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Note that $ln\lambda = -\infty$ corresponds to a model **with no regularisation**. λ reduces overfitting by **reducing the variance of the model parameter** (doesn't mean all coefficients will be nearly equal), but **at expense of adding bias to the model**.

Hence, **increasing λ results in less overfitting but greater bias**.



By a given complexity, the regularity of λ against the root-mean-square error (RMS) is:

- **When λ is very small, model is too complex**. Even though the training error is small, the testing error is very large.
- **When λ is very large, model is too simple**. Both the training error and the testing error are very high.
- At the middle range, as λ increases, **the training error increases but the testing error decreases**, they both approach to a plateau where they remain relatively stable.

- **Hold-out validation v.s K-fold Cross-validation**
  - **Hold-out validation** (Hold-out):
    - Partitioning the available data into 2 parts:

        - The **training set** used to determine the coefficients
        - and **validation set** (hold-out set) used to optimize the model complexity.
    - **Too wasteful**: Data used for training are valuable.
- **K-fold Cross-validation**

  (Leave-one-out is a special case where K equal to the total number of data points):

    - Partitioning the available data into K groups:
        - $K-1$ groups are used to train the model
        - And then use all of the data to assess performance
        - This procedure is repeated K times to ensure **all samples are used** for both **training** ($K-1$ **times**) and **validation** (**only once**)
        - The **average** of the obtained validation errors is used as **an estimation of the testing errors**.
    - **Computationally expensive**:
        - The number of training runs that must be performed is increased by a factor of K.
        - Inefficient for multiple complexity parameters for a single model. If we have several regualarization parameters, exploring combinations of such settings could require a number of training runs that is exponential in the number of parameters.

3. K-Nearest Neighbours (KNN) classifier

- **KNN**: A simple algorithm that stores all available cases and classifies new cases based on a **similarity measure**

  Pseudocode:

```
// X: training data (totally n data points)
// Y: class labels of X
// s: unknown sample
Classify(X, Y, s)
for i = 1 to n do:
    Compute distance *d(**X_i**, s)*
end for
Sort the result in ascending order
find k data points having the smallest distance to s
perform majority vote to give s label
return s label
```

- **Algorithm**: A case is classified by a **majority vote** of its neighbors, with the case being **assigned to the class most common amongst its K nearest neighbors** measured by a **distance function**.
    - **Distance function**:

(For continuous variables)

- **Euclidean** distance: $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$
- **Manhattan** distance: $\sum_{i=1}^{k}|x_i - y_i|$
- **Minkowski** distance: $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{\frac{1}{q}}$

(For categorical variables)

- **Hamming** distance: $D_H = \sum_{i=1}^{k}|x_i - y_i|$, where $x = y \Rightarrow D_H = 0$ otherwise $D_H = 1$

- **Optimal value of** K: Cross-validation is a way to retrospectively determine a good K value by using an independent dataset to validate the K value.
- **Drawback**: Calculating distance measures directly from the training set is in the case where variables have **different measurement scales**, or there is **a mixture of numerical and categorical variables**.
- **Solution**:
  - **Standardize** the training set
  - KNN is **sensitive to outliers** hence remove them before using the algorithm.