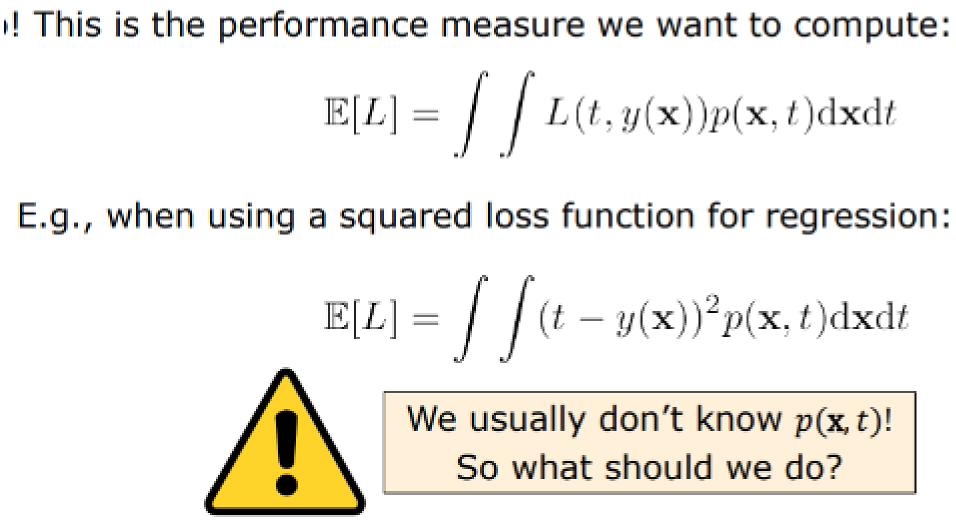
Model Evaluation, Selection and Ensembles

How to evaluate and select models in ML.

Bias Varinance

It is a theoretical framework that allow to have a deeper understanding of why you have some error in a ML model.



! This is the performance measure we want to compute:

$$\mathbb{E}[L] = \int\int L(t, y(\mathbf{x}))p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t$$

E.g., when using a squared loss function for regression:

$$\mathbb{E}[L] = \int\int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t$$

We usually don't know $p(\mathbf{x}, t)$!
So what should we do?

This is the average error the model is going to have for every input. Much more complete assesement of the model than the loss function as the loss function is only based on the dataset and not on all the possible inputs. The problem reside in the fact that the Bias Variance is a theoretical process.

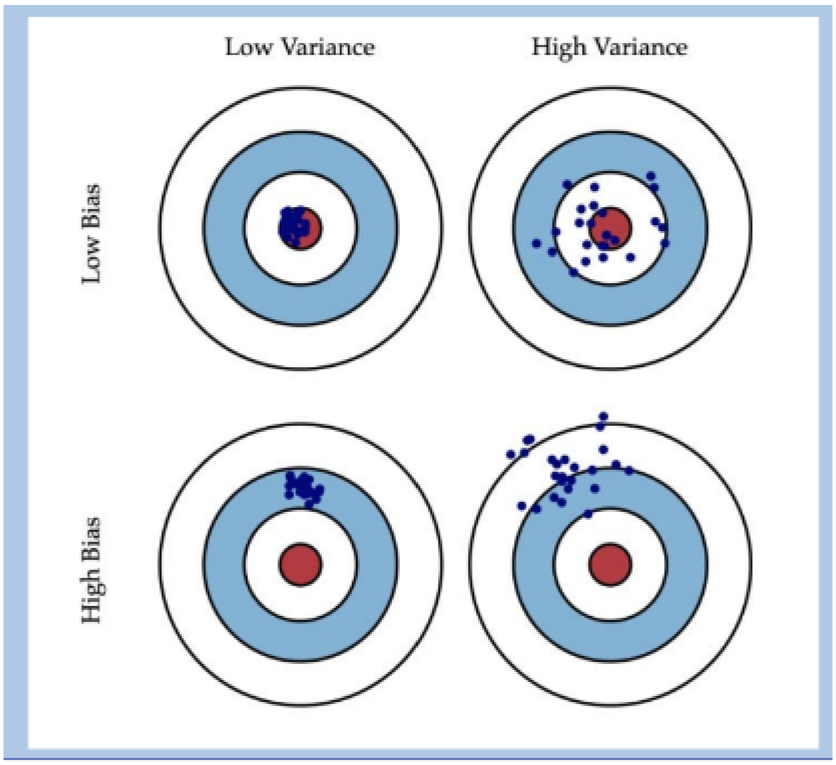
Bias-Variance Decomposition

- The Bias-Variance is a framework to analyze the performance of models
- Definitions and assumptions
  - Data: $t_i = f(\mathbf{x}_i) + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and $Var[\varepsilon] = \sigma^2$
  - Model: $\hat{t_i} = y(\mathbf{x}_i)$ learned from a sampled dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$
  - Performance: $\mathbb{E}\left[(t - y(\mathbf{x}))^2\right]$ (expected square error)
- Thus we can decompose the **expectected square error** as:

$$\begin{aligned}
\mathbb{E}[(t - y(\mathbf{x}))^2] &= \mathbb{E}[t^2 + y(\mathbf{x})^2 - 2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] + \mathbb{E}[y(\mathbf{x})^2] - \mathbb{E}[2ty(\mathbf{x})] \\
&= \mathbb{E}[t^2] \pm \mathbb{E}[t]^2 + \mathbb{E}[y(\mathbf{x})^2] \pm \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + \mathbb{E}[t]^2 + Var[y(\mathbf{x})] + \mathbb{E}[y(\mathbf{x})]^2 - 2f(\mathbf{x})\mathbb{E}[y(\mathbf{x})] \\
&= Var[t] + Var[y(\mathbf{x})] + (f(\mathbf{x}) - \mathbb{E}[y(\mathbf{x})])^2 \\
&= \underbrace{Var[t]}_{\sigma^2} + \underbrace{Var[y(\mathbf{x})]}_{\text{Variance}} + \underbrace{\mathbb{E}[f(\mathbf{x}) - y(\mathbf{x})]^2}_{\text{Bias}^2}
\end{aligned}$$

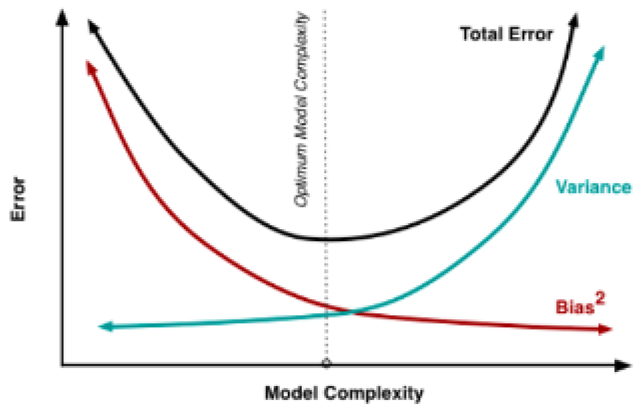
This result can be generalized also for classification.
The function f(.) is generating data. $E[(t - y(\mathbf{x}))^2]$(expected error) is representing all the possible model the neural network will give and for every dataset it will receive.
The higher the σ^2 the higher the noise and the error. Data noise(σ^2) is the variance of data and does not depend neither from data sampling nor from the model complexity.

Model Variance and Bias

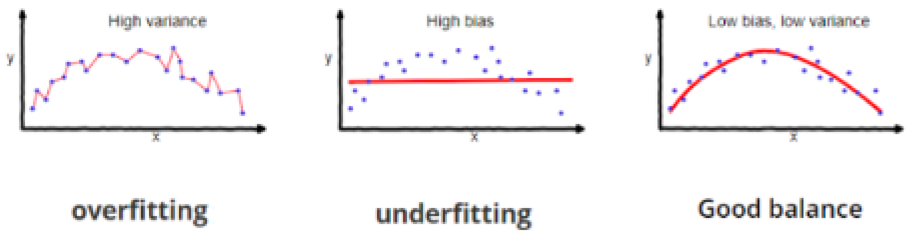


Variance: variance measures the difference between each model learned from a particular dataset and what we expect to learn: $variance = \int E[(y(\mathbf{x}) - E[y(\mathbf{x})])^2]p(\mathbf{x})d\mathbf{x}$
Decreases with simpler models, decreases with more samples
A higher variance lead to a model more sensitive to the dataset used to train it.
The variance contains a term related to the dimension of the dataset. The larger the dataset the smaller the variance and so the error(More data means the capability to handle more complex data).
Bias: bias measures the difference between truth(f) and what we expect to learn($E[y(\mathbf{x})]$): $bias^2 = \int(f(\mathbf{x}) - \bar{y}(\mathbf{x}))^2 p(\mathbf{x})d\mathbf{x}$.
Decreases with more complex models.
It is like a systematic error the model is doing

Credit: http://scott.fortmann-roe.com/docs/BiasVariance.html



Credit: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229



We want a smooth function to have a greater probability of having a smaller variance.

Model Assessment in Practice

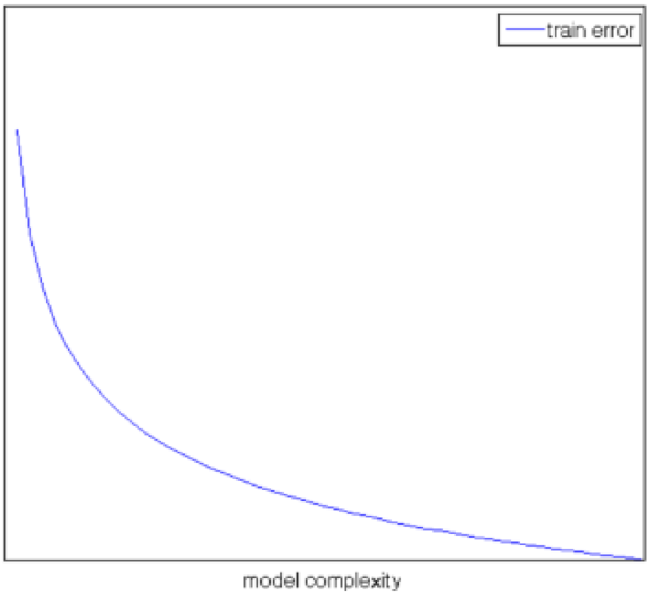
Training Error

- Given $\mathcal{D} = \{\mathbf{x}_i, t_i\}$ with $i = 1, \dots, N$
- We can select a model based on the loss function L computed on $\mathcal{D}$:
  - Regression

$$L_{train} = \frac{1}{N} \sum_{n=1}^{N} (t_n - y(\mathbf{x}_n))^2$$

  - Classification

$$L_{train} = \frac{1}{N} \sum_{n=1}^{N} (I(t_n \neq y(\mathbf{x}_n)))$$



Prediction Error

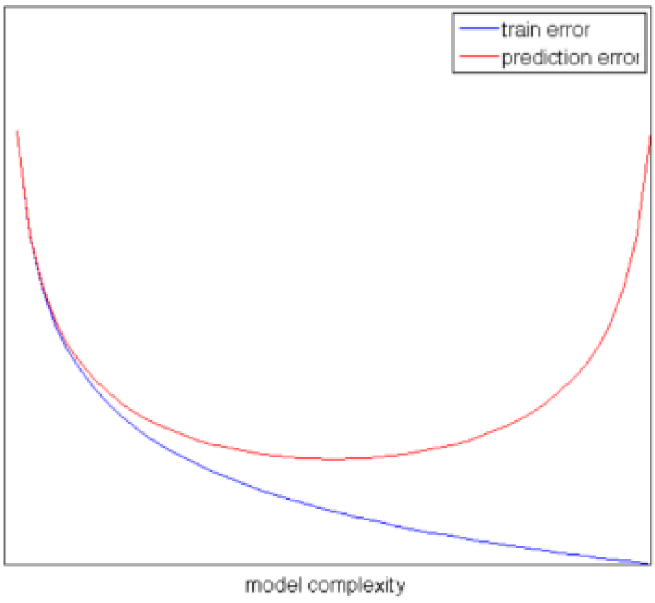
- We already saw that **training error** does not provide a good estimate of the error made on new data, the **prediction error**
  - Regression

$$L_{true} = \int \int (t - y(\mathbf{x}))^2 \, p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t$$

  - Classification

$$L_{true} = \int \int I \, (t \neq y(\mathbf{x})) \, p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t$$

- Unfortunately, we often are not able to model $p(\mathbf{x}, t)$



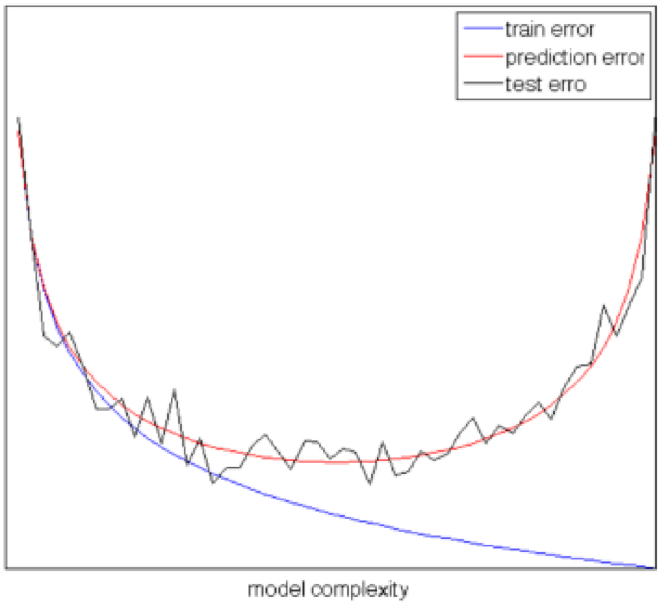
So what to do in practice? Test error

- What can we do in practice?
  - Split randomly data into a **training set** and **test set**
  - Optimize model parameters using the **training set**
  - Estimate the prediction error using the **test set**
    - Regression

$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (t_n - y(\mathbf{x}_n))^2$$

    - Classification

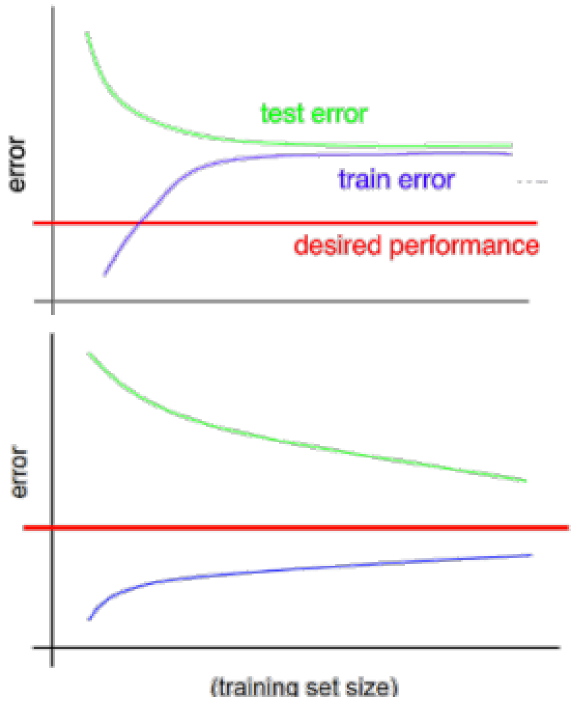
$$L_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (I(t_n \neq y(\mathbf{x}_n)))$$



Increasing the size of the test set will give me a more accurate test error but it will give me less points for the training set and it is a zero gain function. A better solution is to introduce a new set of data called validation set. It will be introduced in a clever way going to cut out only a minimal part of data.

Analysis of train-test error

- The analysis of train-test errors allows to identify possible problems
  - **high bias**: training error is close to test error but they are both higher than expected
  - **high variance**: training error is smaller than expected and it slowly approaches the test error

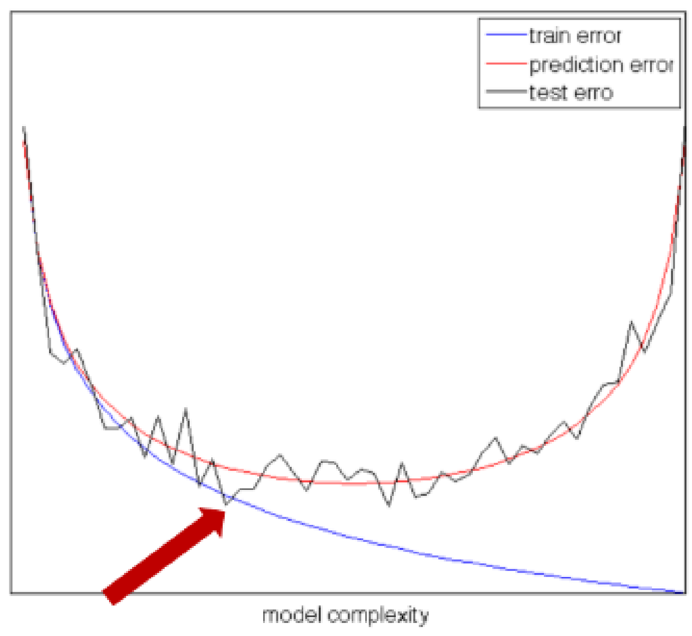


Often data is limited and test error is usually small → prediction error can be either **overestimated** or **underestimated**

Test error cannot be used for model selection → we can **overfit** test set

Only if test set is **never used** for training and model selection, it can provide an **unbiased** estimate of prediction error

So how to perform model selection?



Your ultimate goal in estimating your model is to estimate how good is going to be your model on unvisioned data. The test set is an unbias estimation of the true error of the model but so can be an underestimation or an overestimation of the set. If the model becomes to complex it cannot fit the data, but it also get a large variance(The lower the variance the better for the model). Usually the ideal complexity is founded using the test error and focus on when it stabilise in the range of a value. If I choose the model that optimize this graph i will have a model that underestimate the problem. Another problem is you cannot get out of the estimation problem without an unbiased model as there are no reason to expect to not underestimating/overestimating the problem. I cannot just use a test set and be done with it(overfitting of the test set in choosing the best parameters). A solution consist in the introduction of the Validation Set. It will be used only to select the best model. Then we use this set on the test and the outcome will be the unbiased estimation of the model(practical approach on model selection and unbiased result). There is a small catch: if we have a bunch of data we cut a part and use for training, we cut a part and we use it for test and then we cut the last part for validation, but we are in a limited space so we need to have trade-off. More data available for training the better the model(more complex as the variance is low) but the test and validation set are empirical estimation of the model performance so we want them of a reasonable size to not be useless.

Leave-One-Out Cross Validation (LOO)

I am not forced to split the dataset in two part for the training and validation but I can repeat them multiple time to make them more valid so we take this to the extreme:
given the dataset D(not including test set) for training and validation I remove a single sample of data $\{\mathbf{x}_i, t_i\}$ and train the model on the rest of the data and then compute the error of the resulting model on $\{\mathbf{x}_i, t_i\}$. I repeat this process for each data sample in the data set. At the end I will get an unbiased model for each point in the dataset D finding a way for using the validation set as large as D and using a training set almost as large as D.

$$L_{LOO} = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_{D_i}(\mathbf{x}_i))^2$$

This approach is almost perfect, it provides an almost unbiased estimate of prediction error(slightly pessimistic). The real problem is that this approach is extremely expensive to compute. We use this approach on really small dataset or when the model is inexpensive to train, but in general is difficult to find a real situation to use this approach.

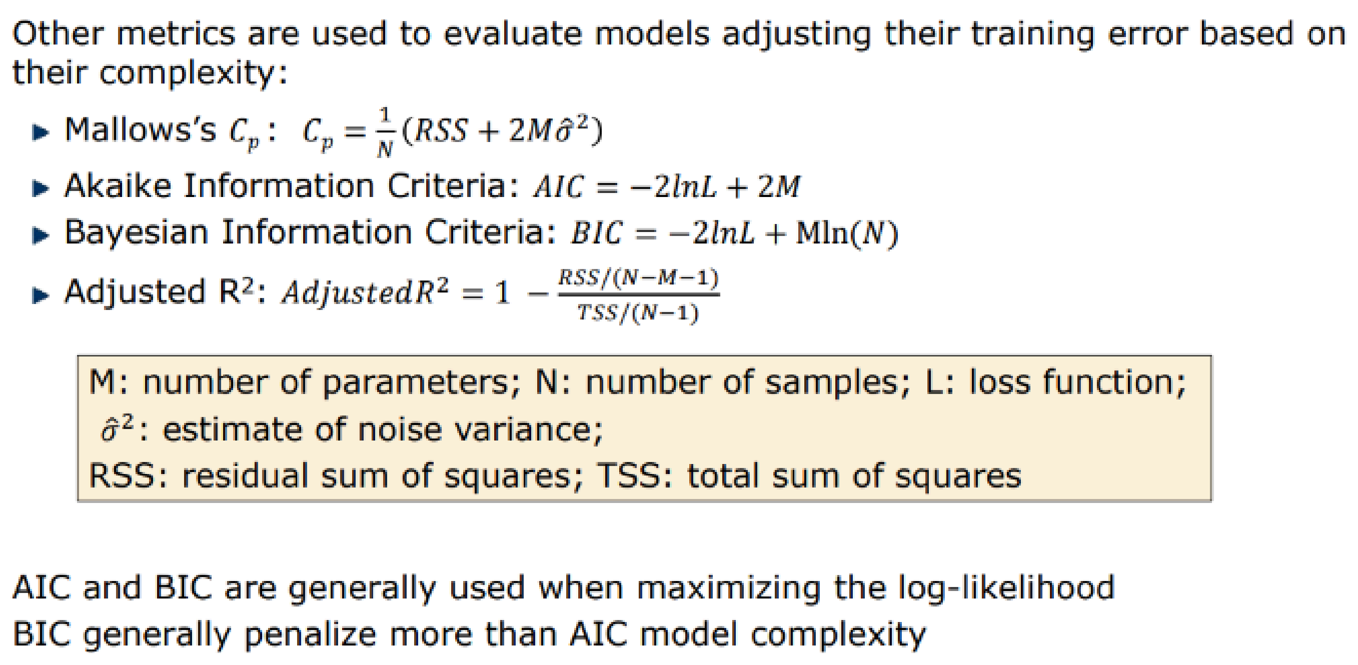
K-Fold Cross Validation

It works with a similar principle but it use batch of data called folds(I randomly split the dataset D in k folds $D_1, \ldots, D_k$). For each fold i train the model on D-{D_i}, compute the error on D_i $L_{D_i} = \frac{k}{N}\sum_{(\mathbf{x}_i,t_i)\in\mathcal{D}_i}(t_n - y_{D/D_i}(\mathbf{x}_i))^2$ and finally estimate the prediction error as the average error computed:

$$L_{k-fold} = \frac{1}{k}\sum_{i=1}^{k} L_{D_i}$$

L_{k-fold} provides a slightly biased estimate of prediction error(pessimistic) but it is much cheaper to compute(usually k=10 is used).
The output of this process is not a model but it will give k version of the model trained on part of the data. It helps choose the best model design among the one proposed. The pipeline of this model is I receive data and select a part for test data, compute k model and then select the best model based on performance of Cross validation and retrain the model on the entirety of the folds and then apply the test set to have the unbiased model. If this result is worse than the ones obtained before the test set you cannot do anything but state that this is the best guess you can deliver.

Complexity-Adjusted Model Evaluation



Other metrics are used to evaluate models adjusting their training error based on their complexity:

- Mallows's C_p: $C_p = \frac{1}{N}(RSS + 2M\hat{\sigma}^2)$
- Akaike Information Criteria: $AIC = -2lnL + 2M$
- Bayesian Information Criteria: $BIC = -2lnL + \mathrm{Mln}(N)$
- Adjusted R²: $AdjustedR^2 = 1 - \frac{RSS/(N-M-1)}{TSS/(N-1)}$

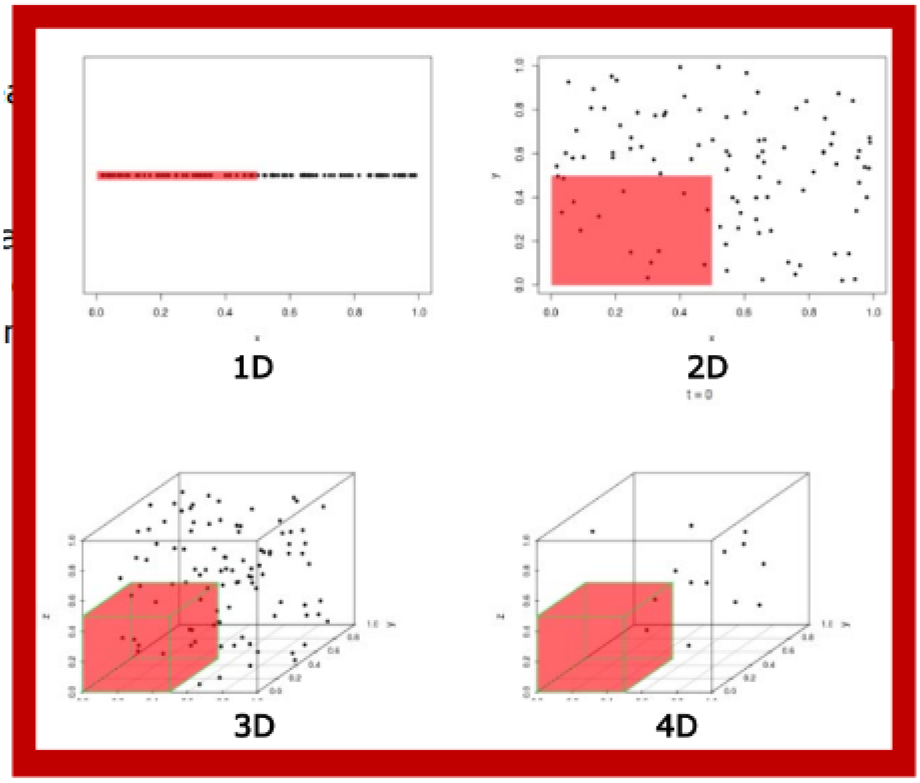
M: number of parameters; N: number of samples; L: loss function; $\hat{\sigma}^2$: estimate of noise variance;
RSS: residual sum of squares; TSS: total sum of squares

AIC and BIC are generally used when maximizing the log-likelihood
BIC generally penalize more than AIC model complexity

These are the last resort, use only if there are no other options. Base decision on training data and add some correction. The idea is that if I cannot do anything other than train the model once I can get an assessment of my model with the adjustment over. The principle is the more complex the model is the less the training error is telling me something useful so I have to penalise more the model.

How to choose the right model complexity?

No really completed answer. We cannot know what is the best model in advance. We can follow some guideline: we need to find a trade off, use mitigating strategies. One major issue is that a low complexity can be solved with a lower complexity model instead a high complexity can be solved by a high complexity model. In a larger space you need much more data to train an equally complexity model in a small space. The higher the dimensionality the higher the dispersion of the same amount of data



Not always using more features is better as it increases the probability of overfitting the data. We can reduce the variance selecting the model with the lowest prediction error. This can be achieved by reducing the variance of the model:

- Feature Selection: we should design the feature space by selecting the most effective subset of all the possible features
- Dimensionality Reduction: the input space can be mapped to a lower-dimensional space
- Regularization: the values of the parameters are shrunked toward zero
  These three approaches are not necessarily mutually exclusive and they can be used toghether

Feature Selection

The simplest idea seems to compare all the possible combinations of the
features, but this is unfeasible.
In practice we use :

- Filter: features are ranked independently based on some evaluation metrics (e.g., correlation, variance, information gain, etc.) and the top k are selected.Very fast but fails to capture any subsest of mutually dependent features
- Embedded: feature selection is performed as a step of the machine learning approach used (e.g., lasso, decision trees, etc.)Not expensive but the features identified are specific to the learning approach
- Wrapper: a search algorithm is used to find a subset of features that are evaluated by training a model with them and assessing its performance. Either a simpler model or a simple machine learning approach can be used to evaluate the features.Greedy algorithms are generally used to search the best subset of features

Dimensionality Reduction

You redesign the features space by finding new way to represent data. The most popular approach is Principal Component Analysis (PCA). The idea in PCA is to find a new basis to represent the data by appling a linear mapping of the previous variable for every new dimension. We design this to be sure that the linear combination capture the maximum level of variability of the data.
This is different from Features Selection because when FS is applied you retain the initial feature meaning. Instead DImensionality Reduction gives new meaning to the features.

Bagging and Boosting

We want to mitigate the trade off between variance and bias combining together various models. If I cannot learn a model with optimal trade off between variance and bias maybe I can obtain it with multiple models.

Bagging

The idea is if I have a random variable this is the prediction of my model and I want to optimize it. The idea is I have N model that I train on N different dataset of the problem and then average them. I pretend of have N dataset but I am doing sample with replacement from the initial dataset. Aggregating the output using an aggregation function depending on the problem I am featuring. This approach violates the assumption of independent dataset. Generally speaking Bagging works well with noisy data and when there is a high variance.

Boosting

It focus on techniques with large bias and low variance. How can we combine low complexity model to create a more complex model but having a still low variance. You train very simple model. The key idea behind boosting is to iteratively train a series of weak learners, with each iteration focusing on the samples that were misclassified in the previous iteration. As final result, an ensemble model is built by combining the outputs of all the weak learners trained. Reduce the bias without varying the variance too much. Works with stable learners. Might have problem with noisy data. It is difficult to parallelise as it is a iterative approach. Can give a boost in performance.