SciML - Machine learning

Mark Asch - IMU/VLP/CSU

2023

Program

Recall machine learning (ML) techniques for scientific applications (see Basic Course for details):

1. General principles of ML.

2. Supervised learning:

   (a) Regression.
   (b) Classification.

3. Unsupervised learning:

   (a) Clustering.
   (b) Trees.

4. Surrogate models and dimensionality reduction.

5. The use of machine learning in scientific applications.

6. The challenges of applying machine learning to scientific applications.

Classes of ML Methods

- supervised learning
- unsupervised learning
- reinforcement learning (AlphaGo)
- self-supervised learning (LLMs, ChatGPT)

Review of ML Methods

Class	Model	Task
Supervised	Linear regression	R
	CART (trees)	R&C
	SVM	R&C
	NN	R&C
	k-NN	C
	Naive Bayes	C
Unsupervised	k-means	clustering
	hierarchical	clustering
	PCA	patterns

- R = regression, C = classification
- CART = Classification and Regression Trees
- SVM = Support Vector Machine
- NN = Neural Network
- PCA = Principal Component Analysis

SUPERVISED LEARNING

Linear Regression

Definition 1 (Regression)**.** Regression is an method of inferential statistics, where we draw evidence and conclusions from measured data concerning complex, realistic, noisy systems. The goal is to learn about the relationship between covariates (or predictors) of interest and response variables (or outcomes).

- Recall: the mathematical framework

- Suppose that we have:

  $\Rightarrow$ a response variable (to explain), Y,
  $\Rightarrow$ p explanatory[1] variables, $X = (X_1, X_2, \ldots, X_p)$,
  $\Rightarrow$ a relationship between Y and X of the form

$$Y = f(X) + \epsilon$$

  $\Rightarrow$ where
    $\rightarrow$ f is an unknown function of $X_1, X_2, \ldots, X_p$

[1] Also called: features, attributes

$\rightarrow$ ϵ is a random error term, independent of X, and with zero mean

- ML is then an ensemble of approaches for estimating f with the objectives of

  $\Rightarrow$ Prediction: $\hat{Y} = \hat{f}(X)$ where $\hat{f}$ is an estimation for f and $\hat{Y}$ is the resulting prediction

  $\Rightarrow$ Inference: to understand how Y varies as a function of X (correlations, importances, linearity, etc.)

- Linear regression is a special case, where we suppose that the response, Y, depends linearly on the unknown coefficients. In other words, we suppose that

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

- Linear regression is the workhorse of statistics!

- Note that LR is linear in the coeffcients, not in the predictors that can have completely general, nonlinear forms.

Linear Regression - Assumptions

- Recall: typical assumptions for a (S)LR model are
  - ⇒ **A1**: The model is linear: e.g. $y_i = f(x;\beta) + \epsilon_i = \beta_0 + \beta_1 x_1 + \epsilon_i$
  - ⇒ **A2**: Unbiased errors: $\mathbb{E}[\epsilon_i|x_i] = \mathbb{E}[\epsilon_i] = 0$
  - ⇒ **A3**: Uncorrelated errors: $\mathrm{Cov}(\epsilon_i, \epsilon_j) = 0$ for $i \neq j$.
  - ⇒ **A4**: Constant variance: $\mathrm{Var}[y_i|x_i] = \sigma^2$
  - ⇒ **A5**: Probability distribution: e.g. $\epsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$.
  - ⇒ **A6**: Representative sampling that generalizes to the population—see Bias and Ethics Lectures.

Linear Regression - Popularity

- Easy to implement (just matrix linear algebra) and fast to compute.
- Solid theoretical foundation providing full array of diagnostic tools.
- Straightforward interpretations—-we obtain an explicit formula!
- Surprisingly flexible and adaptable, though may require some empirical modelling, prior knowledge, experience.
- Provides good approximations in many cases.

Multiple Linear Regression

- Linear regression is a fundamental statistical method used for modeling the relationship between a dependent variable (often denoted as "y") and one or more independent variables (often denoted as "X").

- It assumes that this relationship can be approximated by a linear equation of the form

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{X}_1 + \beta_2 \mathbf{X}_2 + \ldots + \beta_p \mathbf{X}_p + \boldsymbol{\epsilon}$$

  where

  $\Rightarrow$ $\mathbf{y} \in \mathbb{R}^n$ is the dependent variable (the one you want to predict), containing n observations, $y_1, \ldots, y_n$

  $\Rightarrow$ $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_p$ are the independent variables or features, with each $\mathbf{X}_j \in \mathbb{R}^n$ for $j = 1, \ldots, p$.

  $\Rightarrow$ $\beta_0, \beta_1, \ldots, \beta_p$ are the coefficients of the linear equation that need to be estimated.

  $\Rightarrow$ $\boldsymbol{\epsilon} \in \mathbb{R}^n$ represents the error term, which accounts for the variability in $\mathbf{y}$ that cannot be explained by the linear relationship with the $\mathbf{X}$ variables.

- In matrix-vector form, this equation can be written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where:

$\Rightarrow$ $\mathbf{y}$ is a column vector with dimension $n \times 1$ of the dependent variable values.

$\Rightarrow$ $\mathbf{X}$ is the design matrix , of dimension $n \times (p+1)$ where each row represents one observation and each column represents an independent variable.

$\Rightarrow$ $\boldsymbol{\beta}$ is a column vector, dimension $(p+1) \times 1$, of the coefficient estimates, including β_0

$\Rightarrow$ $\boldsymbol{\epsilon}$ is IID, centered Gaussian[2] noise with $\mathrm{E}(\boldsymbol{\epsilon}) = 0$ and $\mathrm{Cov}(\boldsymbol{\epsilon}) = \sigma^2\mathbf{I}$, sometimes written as $\epsilon_i \overset{\text{iid}}{\sim} (0, \sigma^2)$

[2]The Gaussian noise model is NOT restrictive, but is the most arbitrary one with the least number of hypotheses. We only assume centrality and decreasing probability of large and small values.

- Writing this out, we obtain

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ \vdots & \vdots & x_{ij} & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{bmatrix},$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_p \end{bmatrix}, \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

  or in the SLR case,

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}}_{=\mathbf{X}} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}}_{\boldsymbol{\beta}} + \underbrace{\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}}_{+\boldsymbol{\epsilon}}$$

- In multiple linear regression, the coefficients (also known as the parameter estimates) are computed, as in simple linear regression, using a method called ordinary least squares (OLS).

⇒ The goal of OLS is to minimize the sum of squared errors (SSE), which is the sum of the squared differences between the observed $\mathbf{y}$ values and the predicted $\mathbf{y}$ values.

⇒ Mathematically, this is represented as,

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the predicted value of y for the i-th observation.

- The optimal solution of this minimization problem can in fact be derived explicitly, and is quite easily shown to be given by

$$\boldsymbol{\beta}_{\text{LS}} = \left(\mathbf{X}^{\text{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\text{T}}\mathbf{y}$$

Recall: Noise Model

- A noise model is essential for any analysis, particularly Bayesian (of which regression is an example)

- Neutrality of the Gaussian hypothesis for the noise model:

  $\Rightarrow$ the noise is centered : its mean value is zero, but it can take any value (small or big)
  $\Rightarrow$ large amplitudes/deviations are less and less probables: the variance is finite
  $\Rightarrow$ independence between observations (otherwise, this is part of the trends of the model)

- We neither suppose, nor impose that the noise really follows a Gaussian law...

Linear Regression: Diagnostics

- The only ML method with complete diagnostics:

  $\Rightarrow$ Coefficient of determination (R-squared)

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} = \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{TSS}}$$

  represents the proportion of the variance "explained" by the model. Beware of the limitations of this coefficient!

  $\Rightarrow$ Mean-squared Error:

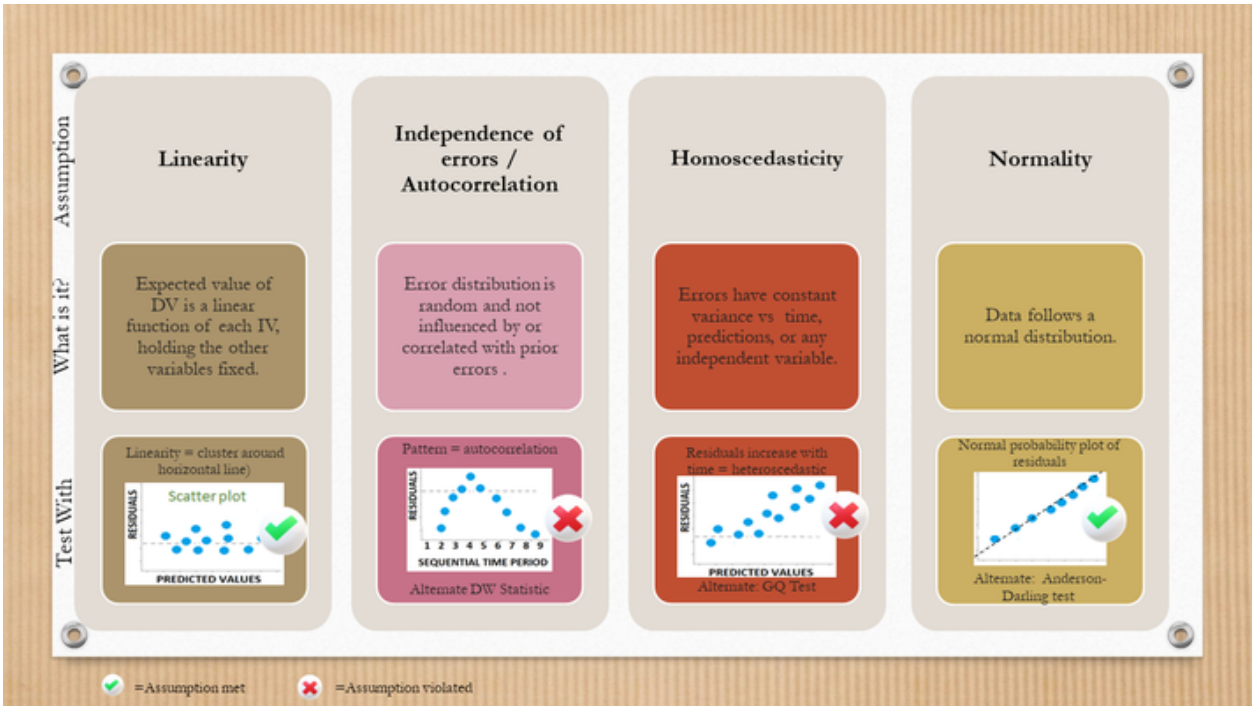
$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 .$$

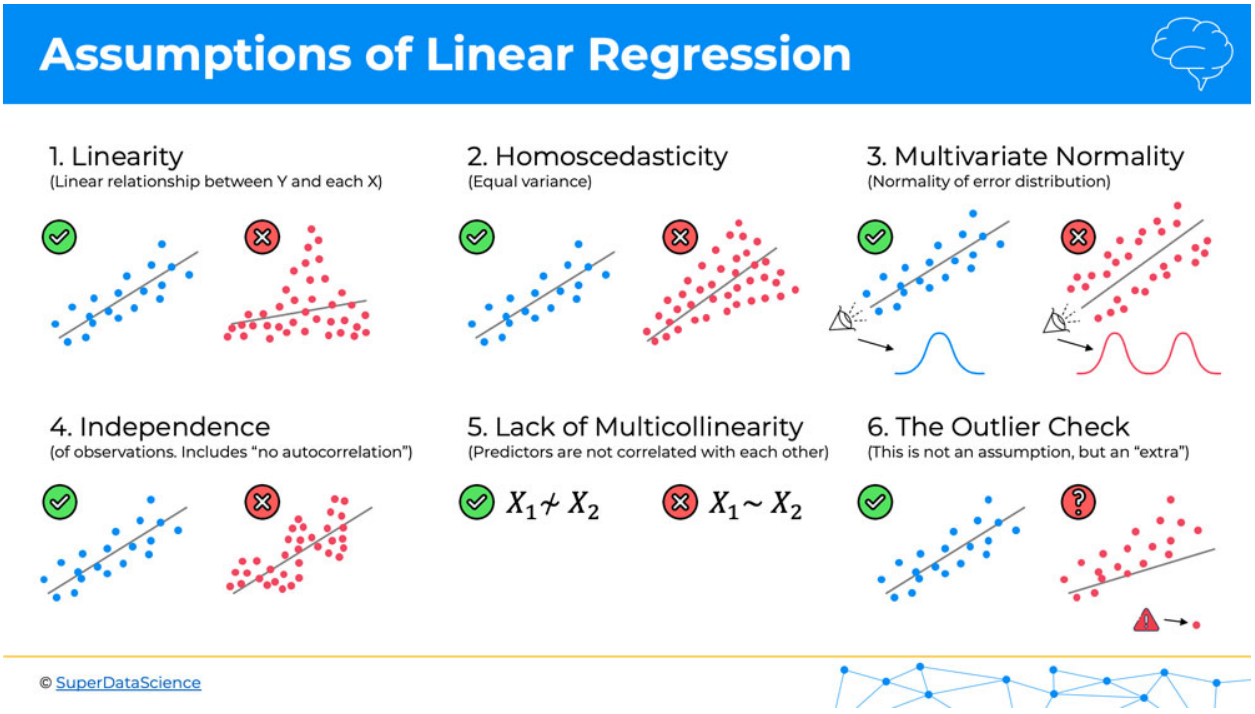
  Can be more reliable than R-squared, but is context-dependent (not between 0 and 1).

  $\Rightarrow$ p-values and significance: these are no longer considered to be advisable, and in general should not be used.

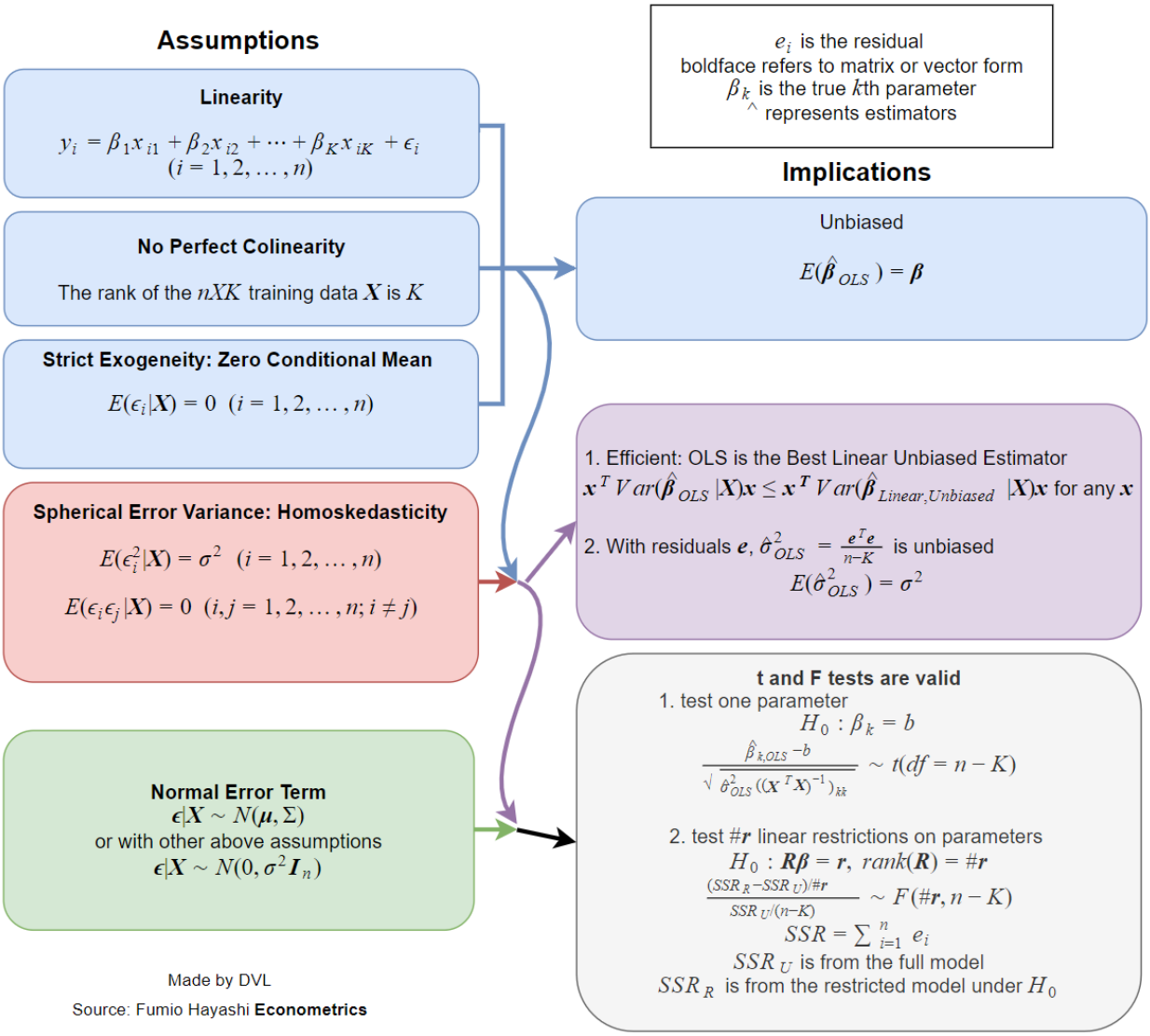
- Graphical diagnostics

1. The plots ("Residuals vs Fitted" and "Scale-Location") should not give any clear trends (they should neither be all increasing or all decreasing). This shows:
   (a) that on average, the regression line is well fitted to the data, and thus the hypothesis of linearity is acceptable,
   (b) that the variance is constant and of the same value for all the observations.
2. The "normal Q-Q" plot should show points distributed around the dashed line and that follow the line approximately, without marked deviations (especially at the extremities). This shows that the hypothesis of the residues having a normal distribution, is satisfied.
3. The last "Cook distances", should not show any point that exceeds 1 on the abscissa. This shows the presence of influential data.


Assumption
What is it?
Test With
Linearity
Expected value of DV is a linear function of each IV, holding the other variables fixed.
Linearity = cluster around horizontal line)
Scatter plot
RESIDUALS
PREDICTED VALUES
Independence of errors / Autocorrelation
Error distribution is random and not influenced by or correlated with prior errors .
Pattern = autocorrelation
RESIDUALS
1 2 3 4 5 6 7 8 9
SEQUENTIAL TIME PERIOD
Alternate DW Statistic
Homoscedasticity
Errors have constant variance vs time, predictions, or any independent variable.
Residuals increase with time = heteroscedastic
RESIDUALS
PREDICTED VALUES
Alternate: GQ Test
Normality
Data follows a normal distribution.
Normal probability plot of residuals
Alternate: Anderson-Darling test
=Assumption met
=Assumption violated


Assumptions of Linear Regression
1. Linearity
(Linear relationship between Y and each X)
2. Homoscedasticity
(Equal variance)
3. Multivariate Normality
(Normality of error distribution)
4. Independence
(of observations. Includes "no autocorrelation")
5. Lack of Multicollinearity
(Predictors are not correlated with each other)
$X_1 \not\sim X_2$
$X_1 \sim X_2$
6. The Outlier Check
(This is not an assumption, but an "extra")
© SuperDataScience

Finite Sample OLS



Recall: Simple Linear Regression Model

- Linear Function: The mean of the response, $\mathrm{E}(Y_i)$, at each value of the predictor, x_i, is a Linear function of the x_i.
- Independent: The errors, ϵ_i, are Independent.
- Normally Distributed: The errors, ϵ_i, at each value of the predictor, x_i, are Normally distributed.
- Equal variances (denoted σ^2): The errors, ϵ_i, at each value of the predictor, x_i, have Equal variances (denoted σ^2).

Regression: Shrinkage and Selection

- **Prediction**:
  - ⇒ Linear regression: $E(Y_j|X) = X\beta$;
  - ⇒ Or for a more general regression function: $E(Y_j|X) = f(X)$;
  - ⇒ In a prediction context, there is less concern about the values of the components of the right-hand side, rather interest is on the total contribution.

- **Variable Selection**:
  - ⇒ The driving force behind variable selection:
    - → The desire for a parsimonious regression model (one that is simpler and easier to interpret);T
    - → he need for greater accuracy in prediction.
  - ⇒ The notion of what makes a variable "important" is still not well understood, but one interpretation (Breiman, 2001) is that a variable is important if dropping it seriously affects prediction accuracy.
  - ⇒ Selecting variables in regression models is a complicated problem, and there are many conflicting views

on which type of variable selection procedure is best, e.g. LRT, F-test, AIC, and BIC.

- There are two main types of **stepwise procedures** in regression:
  - ⇒ Backward elimination: eliminate the least important variable from the selected ones.
  - ⇒ Forward selection: add the most important variable from the remaining ones.
  - ⇒ A hybrid version that incorporates ideas from both main types: alternates backward and forward steps, and stops when all variables have either been retained for inclusion or removed.

- **Criticisms** of Stepwise Methods:
  - ⇒ There is no guarantee that the subsets obtained from stepwise procedures will contain the same variables or even be the "best" subset.
  - ⇒ When there are more variables than observations ($p > n$), backward elimination is typically not a feasible procedure.
  - ⇒ The maximum or minimum of a set of correlated F statistics is not itself an F statistic.

⇒ It produces a single answer (a very specific subset) to the variable selection problem, although several different subsets may be equally good for regression purposes.

⇒ The computing is easy by the use of R function `step()` or `regsubsets()`. However, to specify a practically good answer, you must know the practical context in which your inference will be used.

Linear Regression: LASSO and Ridge

- Ridge regression and Lasso regression are two popular regularization techniques used in linear regression to address issues such as multicollinearity, feature selection and overfitting.

Motivation: ill-conditioned X

- Because the LS estimates depend upon $(X^{\mathrm{T}}X)^{-1}$, we would have problems in computing β_{LS} if $X^{\mathrm{T}}X$ were singular or nearly singular.

- In those cases, small changes to the elements of X lead to large changes in $(X^{\mathrm{T}}X)^{-1}$,

- The least square estimator β_{LS} i may provide a good fit to the training data, but it will not fit sufficiently well to the test data.

Linear Regression: Ridge

Objective:

- Ridge regression, also known as L2 regularization, aims to prevent multicollinearity by adding a penalty term to the linear regression objective function.

- The objective is to find the values of coefficients $\boldsymbol{\beta}$ that minimize the following modified sum of squared errors,

$$\text{SSE}_{\text{Ridge}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where

$\Rightarrow$ the second term, $\lambda \sum_{j=1}^{p} \beta_j^2$, is the regularization term.

$\Rightarrow$ λ (lambda) is the regularization parameter, which controls the strength of regularization

$\Rightarrow$ this is a hyperparameter and needs to be tuned:

→ If we choose $\lambda = 0$, we have p parameters (since there is no penalization).
→ If λ is large, the parameters are strongly constrained and the degrees of freedom will effectively be lower, tending to 0 as $\lambda \to \infty$.
→ We need to choose a value inbetween...

Effect:

1. Ridge regression shrinks the coefficients towards zero but doesn't eliminate them entirely.

2. It is effective in preventing multicollinearity by encouraging smaller coefficient values.

Linear Regression: LASSO

Objective:

- Lasso regression, also known as L1 regularization, not only prevents multicollinearity but also performs feature selection by adding a penalty term to the linear regression objective function.

- The objective is to find the values of coefficients β that minimize the following modified sum of squared errors,

$$\text{SSE}_{\text{Lasso}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

where

$\Rightarrow$ $\text{SSE}_{\text{Lasso}}$ is the sum of squared errors with a regularization term.

$\Rightarrow$ The second term, $\lambda \sum_{j=1}^{p} |\beta_j|$, is the regularization term with the absolute values of coefficients.

Effect:

1. Lasso regression encourages sparsity, meaning it tends to force some coefficients to be exactly zero, effectively removing irrelevant features from the model.

2. It is useful when you have a large number of features and want automatic feature selection. As p increases, the multidimensional diamond, described by the L1 norm, has an increasing number of corners, and so it is highly likely that some coefficients will be set equal to zero. Hence, the lasso performs shrinkage and (effectively) subset selection.

Regression - Generalizations

- generalized linear models GLM
- categorical predictors
- spline regressions (MARS)

Classification—simple/linear

Note

Please review this material in the Basic Course.

- k-NN:

  $\Rightarrow$ The only parameter that can adjust the complexity of k-NN is the number of neighbors k.

  $\Rightarrow$ The larger k is, the smoother the classification boundary.

  $\Rightarrow$ Or we can think of the complexity of k-NN as lower when k increases.

- Logistic, Naive Bayes, LDA

- nonlinear methods—see below

Applyng k-NN in Practice

If you wish to apply k-NN in practice proceed as follows:

1. Preprocess your data: Normalize the features in your data (e.g. one pixel in images) to have zero mean and unit variance. We will cover this in more detail in later sections, and chose not to cover data normalization in this section because pixels in images are usually homogeneous and do not exhibit widely different distributions, alleviating the need for data normalization.

2. If your data is very high-dimensional, consider using a dimensionality reduction technique such as PCA, NCA, or even Random Projections.

3. Split your training data randomly into train/val splits.
   (a) As a rule of thumb, between 70-90% of your data usually goes to the train split.
   (b) This setting depends on how many hyperparameters you have and how much of an influence you expect them to have.

(c) If there are many hyperparameters to estimate, you should err on the side of having larger validation set to estimate them effectively.
(d) If you are concerned about the size of your validation data, it is best to split the training data into folds and perform cross-validation.
(e) If you can afford the computational budget it is always safer to go with cross-validation (the more folds the better, but more expensive).

4. Train and evaluate the k-NN classifier on the validation data (for all folds, if doing cross-validation)

   (a) for many choices of k (e.g. the more the better) and
   (b) across different distance types (L1 and L2 are good candidates)

5. If your k-NN classifier is running too long, consider using an Approximate Nearest Neighbor library (e.g. FLANN) to accelerate the retrieval (at cost of some accuracy).

6. Take note of the hyperparameters that gave the best results.

(a) There is a question of whether you should use the full training set with the best hyperparameters, since the optimal hyperparameters might change if you were to fold the validation data into your training set (since the size of the data would be larger).
(b) In practice it is cleaner to not use the validation data in the final classifier and consider it to be burned on estimating the hyperparameters. Evaluate the best model on the test set.
(c) Report the test set accuracy and declare the result to be the performance of the k-NN classifier on your data.

Support Vector Machines

- The support vector machine (SVM) algorithm performs supervised classification.
- It can provide excellent performance in a broad range of contexts.
- It is considered to be one of the foremost "black box" methods of statistical learning—see Bias and Ethics Lectures.
- The basic property of the algorithm is its generalization of the boundary between classes to nonlinear curves and hypersurfaces.
- Please refer to the Basic Course for full explanation and details.

SVM in a nutshell

- The SVM algorithm looks for a linearly separable hyperplane, or a decision boundary separating members of one class from the other.

  $\Rightarrow$ If such a hyperplane exists, the work is done!
  $\Rightarrow$ If such a hyperplane does not exist, SVM uses a nonlinear mapping to transform the training data into a higher dimension.

- Then it searches for the linear optimal separating hyperplane.

  $\Rightarrow$ With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane.
  $\Rightarrow$ The SVM algorithm finds this hyperplane using support vectors and margins.

- As a training algorithm, SVM may not be very fast compared to some other classification methods, but owing

to its ability to model complex nonlinear boundaries, SVM has high accuracy.

- SVM is comparatively less prone to overfitting.

- SVM has successfully been applied to handwritten digit recognition, text classification, speaker identification etc.

SVM for Regression

- We can use SVM to perform regressions too.
- Instead of computing the coefficients β_i to minimize the SLR least squares criterion on the residuals, SVM imposes a modified loss.
- This loss uses only residuals that are larger in absolute value than a given positive constant.
- The loss function is defined as

$$L(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon, \\ |y - \hat{y}| - \epsilon & \text{otherwise,} \end{cases}$$

  implying that any point lying within an ϵ-neighborhood—or tube—around the prediction is not penalized.
- This is related to LASSO regression presented above.

- This gives an advantage over classical least squares regression, in that we can fit extremely nonlinear functions, such as those that exhibit saturation and thresholding.

Decision Trees and Random Forests

- **Background**:
  - ⇒ Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.
  - ⇒ Learned trees can also be re-represented as sets of if-then rules to improve human readability.
  - ⇒ These learning methods are among the most popular of inductive inference algorithms
  - ⇒ They have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.
  - ⇒ Classification trees are a hierarchical way of partitioning the space. We start with the entire space and recursively divide it into smaller regions. In the end, every region is assigned to a class label.

- **Recall**:
  - ⇒ Decision trees can be applied to both classification and regression problems.

⇒ They are particularly clear to interpret, but suffer from a lack of robustness.

⇒ Random Forests (RF), bagging and boosting repair the robustness and are extremely good classifiers and regressors.

- To go further, we need to study:

⇒ pruning - usually done by cross-validation

⇒ bagging

⇒ boosting

⇒ RF and variable importance for model reduction/feature selection.

Random Forests

- Bagging—bootstrap aggregation—constructs a large number of trees with bootstrap samples from a dataset.

- For RF, as each tree is constructed, take a random sample of predictors before each node is split.

  ⇒ For example, if there are twenty predictors, choose a random five as candidates for constructing the best split.

  ⇒ Repeat this process for each node until the tree is large enough. And as in bagging, do not prune.

- Why do RFs work?

  ⇒ Variance reduction:

  → The trees are more independent because of the combination of bootstrap samples and random draws of predictors.

  → It is apparent that random forests are a form of bagging, and the averaging over trees can substantially reduce instability that might otherwise result.

Moreover, by working with a random sample of predictors at each possible split, the fitted values across trees are more independent. Consequently, the gains from averaging over a large number of trees (variance reduction) can be more dramatic.

⇒ Bias reduction:

→ A very large number of predictors can be considered, and local feature predictors can play a role in tree construction.

→ Random forests are able to work with a very large number of predictors, even with more predictors than there are observations. An obvious gain with random forests is that more information may be brought to reduce bias of fitted values and estimated splits.

→ There are often a few predictors that dominate the decision tree fitting process because on the average they consistently perform just a bit better than their competitors. Consequently, many other predictors, which could be useful for very local features of the data, are rarely selected as splitting variables. With random forests computed for a large enough number of trees, each predictor

will have at least several opportunities to be the predictor defining a split. In those opportunities, it will have very few competitors. Much of the time a dominant predictor will not be included. Therefore, local feature predictors will have the opportunity to define a split.

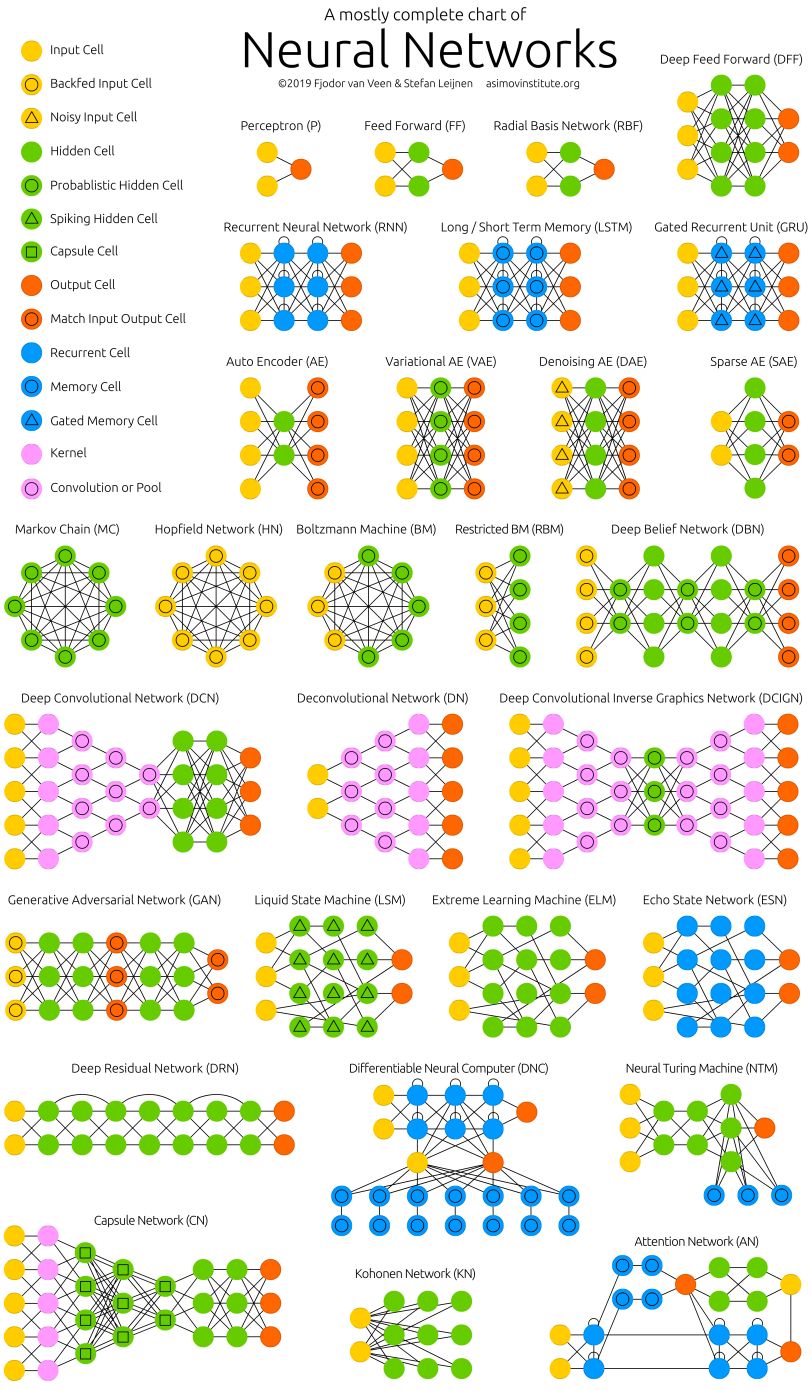
- Indeed, random forests are among the very best classifiers invented to date (Breiman, 2001).

- Random forests include 3 main tuning parameters.

  1. Node Size: unlike in decision trees, the number of observations in the terminal nodes of each tree of the forest can be very small. The goal is to grow trees with as little bias as possible.
  2. Number of Trees: in practice, 500 trees is often a good choice.
  3. Number of Predictors Sampled: the number of predictors sampled at each split would seem to be a key tuning parameter that should affect how well random forests perform. Sampling 2-5 each time is often adequate.

SUPERVISED LEARNING - Neural Networks

Neural Networks



- Universal Approximation Property–important for PINN and DeepONet (see PINN Lecture)

- Limits of universality:

  $\Rightarrow$ you may need to represent an exponentially complex network

  $\Rightarrow$ if you can learn any function, then you WILL certainly overfit

NN - background and motivation

- Previously (see above and in Basic Course), we discussed linear models for regression and classification.

  $\Rightarrow$ In particular, logistic regression, that in the binary case, corresponds to the model

$$p(y|x, w) = \mathrm{Ber}(y|\sigma(w^T x)),$$

  $\Rightarrow$ linear regression, corresponds to the Gaussian model

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

  $\Rightarrow$ generalized linear models (GLM), that generalize these models to other kinds of output distributions, such as Poisson.

  $\Rightarrow$ However, all these models make the strong assumption that the input-output mapping is linear.

- A simple way of increasing the flexibility of such models is to perform a feature transformation, by replacing x with $\varphi(x)$.

⇒ For example, we can use a polynomial transform, which in 1D is given by $\varphi(x) = [1, x, x^2, x^3, ...]$. This is sometimes called basis function expansion.

⇒ The model now becomes

$$f(x; \theta) = W\varphi(x) + b$$

⇒ This is still linear in the parameters $\theta = (W, b)$, which makes model fitting easy (since the negative log-likelihood is convex). However, having to specify the feature transformation by hand is very limiting.

- A natural extension is to endow the feature extractor with its own parameters, θ_2, to get

$$f(x; \theta) = W\varphi(x; \theta_2) + b$$

where $\theta = (\theta_1, \theta_2)$and $\theta_1 = (W, b)$. We can obviously repeat this process recursively, to create more and more complex functions.

- If we compose L functions, we get

$$f(x; \theta) = f_L(f_{L-1}(\cdots(f_1(x))\cdots))$$

where $f_l(x) = f(x; \theta_l)$ is the function at layer l. This is the key idea behind (deep) neural networks or (D)NNs.

- The term "(D)NN" actually encompasses a larger family of models, in which we compose differentiable functions into any kind of DAG (directed acyclic graph), mapping input to output.

  $\Rightarrow$ Equation above is the simplest example where the DAG is a chain. This is known as a feedforward neural network (FFNN/FCNN) or multilayer perceptron (MLP).
  $\Rightarrow$ An MLP assumes that the input is a fixed-dimensional vector, say $x \in \mathbb{R}^D$.
  $\Rightarrow$ It is common to call such data "structured data" or "tabular data", since the data is often stored in an $N \times D$ design matrix, where each column (feature) has a specific meaning, such as height, weight, age, etc.

- We discuss below other kinds of DNNs that are more suited to "unstructured data" such as images and text, where the input data is variable sized, and each individual

element (e.g., pixel or word) is often meaningless on its own.

⇒ convolutional neural networks (CNN), which are designed to work with images;
⇒ recurrent neural networks (RNN) and transformers, which are designed to work with sequences
⇒ graph neural networks (GNN), which are designed to work with graphs

Differentiable MLPs

- For MLPs, we define differentiable activation functions $\varphi : \mathbb{R} \to R$. More precisely, we define the hidden units z_l at each layer l to be a linear transformation of the hidden units at the previous layer passed element-wise through this activation function,

$$z_l = f_l(z_{l-1}) = \varphi_l(b_l + W_l z_{l-1})$$

  or, in scalar form,

- If we now compose L of these functions together, we can compute the gradient of the output with respect to the parameters in each layer using the chain rule, also known as backpropagation, as we explain in the Lecture on Automatic Differentiation.

- And once we have the gradient, we can invoke a gradient-based optimization algorithm to minimize a loss function, measuring the mismatch between the model and the measurements, as usual.

Activation Functions

- In the early days of neural networks, a common choice was to use a sigmoid (logistic) function, which can be seen as a smooth approximation to the Heaviside function used in a perceptron

$$\sigma(a) = \frac{1}{1 + \mathrm{e}^{-a}}$$

- However, the sigmoid function saturates at 1 for large positive inputs, and at 0 for large negative inputs.

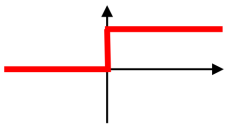
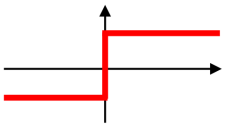
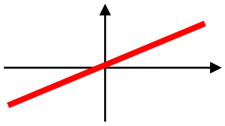
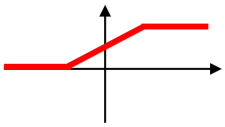
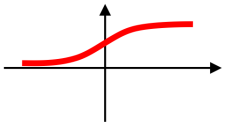
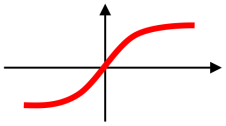
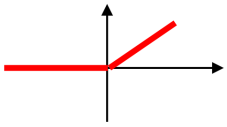
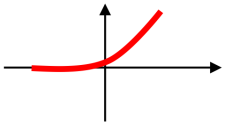
- Another common choice is the tanh function, which has a similar shape, but saturates at -1 and $+1$.

- In the saturated regimes, the gradient of the output wrt the input will be close to zero, so any gradient signal from higher layers will not be able to propagate back to earlier layers. This is called the vanishing gradient problem, and it makes it hard to train the model using gradient descent.

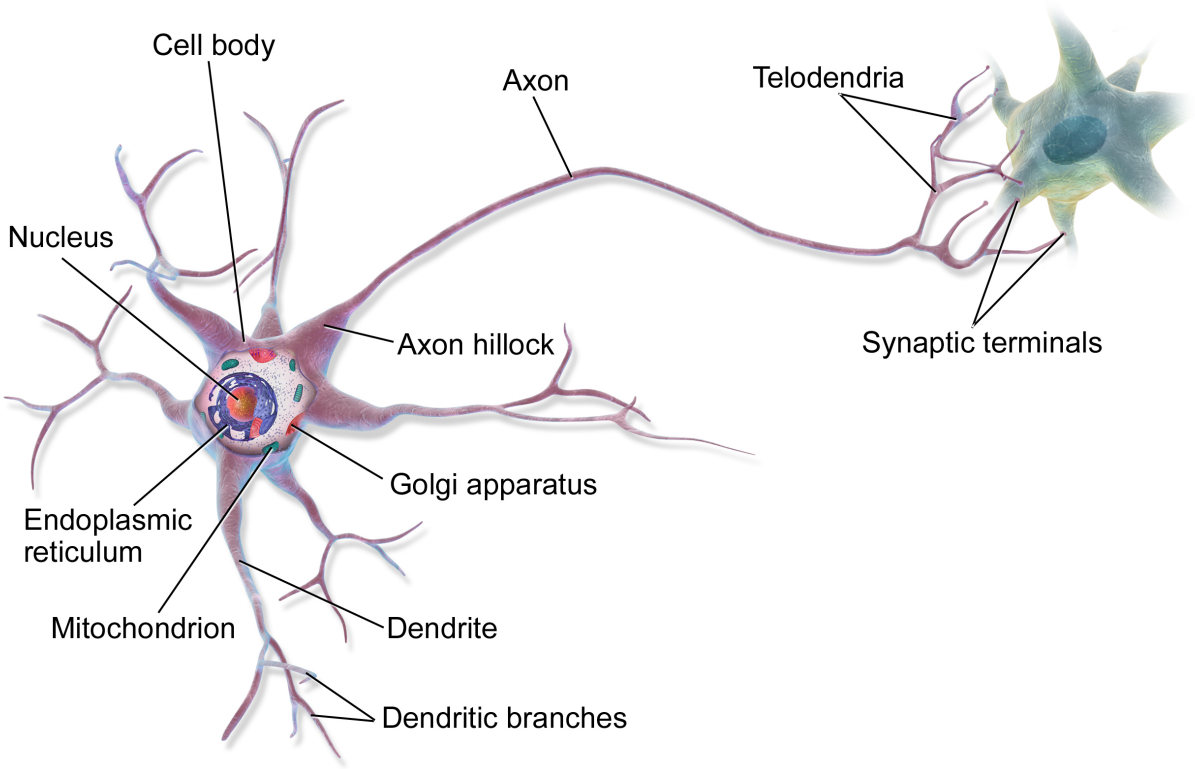
- One of the keys to being able to train very deep models is to use non-saturating activation functions. Several different functions have been proposed. The most common is rectified linear unit or ReLU,

$$\mathrm{ReLU}(a) = \max(a, 0) = a\mathbb{I}, \quad (a > 0)$$

- All of the above, together with the availability of data, open-source software and GPU accelerators, has led to the "deep learning revolution" that started in the 2010's (about a decade ago...)

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

NNs and AI - connections with biology?



- What are the connections between the kinds of neural networks we have discussed above, known as artificial neural networks or ANNs, and real neural networks, as found in humans and in the animal kingdom?

- Let us consider a model of a single neuron...

⇒ To a first approximation, we can say that whether neuron k fires, denoted by $h_k \in \{0, 1\}$, depends on the activity of its inputs, denoted by $x \in \mathbb{R}^D$, as well as the strength of the incoming connections, which we denote by $w_k \in \mathbb{R}^D$.

⇒ We can compute a weighted sum of the inputs using $a_k = w_k^T x$. These weights can be viewed as "wires" connecting the inputs x_d to neuron h_k; these are analogous to dendrites in a real neuron.

⇒ This weighted sum is then compared to a threshold, b_k, and if the activation exceeds the threshold, the neuron fires; this is analogous to the neuron emitting an electrical output or action potential.

⇒ Thus we can model the behavior of the neuron using $h_k(x) = H(w_k^T x - b_k)$, where $H(a) = \mathbb{I}(a > 0)$ is the Heaviside function. This is called the McCulloch-Pitts model of the neuron, and was proposed in 1943...

- We can then combine multiple such neurons together to make an ANN. The result has sometimes been viewed as a model of the brain.

- However, ANNs differ from biological brains in many ways, including the following:

  ⇒ Most ANNs use backpropagation to modify the strength of their connection. However, real brains do not use backprop, since there is no way to send information backwards along an axon. Instead, they use local update rules for adjusting synaptic strengths.

  ⇒ Most ANNs are strictly feedforward, but real brains have many feedback connections. It is believed that this feedback acts like a prior, which can be combined with bottom up likelihoods from the sensory system to compute a posterior over hidden states of the world, which can then be used for optimal decision making.

  ⇒ Most ANNs use simplified neurons consisting of a weighted sum passed through a nonlinearity, but real biological neurons have complex dendritic tree structures (see Figure), with complex spatio-temporal dynamics.

  ⇒ Most ANNs are smaller in size and number of connections than biological brains. Of course, ANNs are getting larger every week, fueled by various new hardware accelerators, such as GPUs and TPUs (ten-

sor processing units), etc. However, even if ANNs match biological brains in terms of number of units, the comparison is misleading since the processing capability of a biological neuron is much higher than an artificial neuron (see point above).

⇒ Most ANNs are designed to model a single function, such as mapping an image to a label, or a sequence of words to another sequence of words. By contrast, biological brains are very complex systems, composed of multiple specialized interacting modules, which implement different kinds of functions or behaviors such as perception, control, memory, language, etc.

- It is commonly believed that the low level details of biological brains do not matter if our goal is to build "intelligent machines", just as aeroplanes do not flap their wings. However, presumably "AIs" will follow similar "laws of intelligence" to intelligent biological agents, just as planes and birds follow the same laws of aerodynamics.

- Unfortunately, we do not yet know what the "laws of intelligence" are, or indeed if there even are such

laws. One approach is to make the assumption that any intelligent agent should follow the basic principles of information processing and Bayesian decision theory, which is known to be the optimal way to make decisions under uncertainty.

- In practice, the optimal Bayesian approach is often computationally intractable. In the natural world, biological agents have evolved various algorithmic "shortcuts" to the optimal solution; this can explain many of the heuristics that people use in everyday reasoning. As the tasks we want our machines to solve become harder, we may be able to gain insights from neuroscience and cognitive science for how to solve such tasks in an approximate way. However, we should also bear in mind that AI/ML systems are increasingly used for safety-critical applications, in which we might want and expect the machine to do better than a human. In such cases, we may want more than just heuristic solutions that often work; instead we may want provably reliable methods, similar to other engineering fields... There is still a long way to go before reaching this point—see Bias and Ethics Lectures.

UNSUPERVISED LEARNING

Clustering

- Evaluating clustering results is inevitably subjective. For supervised classification, we have the true labels. Therefore, we can compute the error rate to assess the result. However, in unsupervised clustering, we do not have the class labels given for the training data.

  $\Rightarrow$ One heuristic generally accepted is that points in the same cluster should be tight and points in different groups should be as far apart as possible.

- k-means:

  $\Rightarrow$ The k-means algorithm reflects the heuristic by attempting to minimize the total within-cluster distances between each data point and its corresponding prototype (or centroid).

- The agglomerative clustering method is also called a bottom-up method as opposed to k-means or k-center methods that are top-down.

⇒ In a top-down method, a data set is divided into more and more clusters.

⇒ In a bottom-up approach, all the data points are treated as individual clusters to start with and gradually merged into bigger and bigger clusters.

- In agglomerative clustering, clusters are generated hierarchically.

  ⇒ We start by taking every data point as a cluster.

  ⇒ Then we merge two clusters at a time.

  ⇒ In order to decide which two clusters to merge, we compare the pairwise distances between any two clusters and pick a pair with the minimum distance. Once we merge two clusters into a bigger one, a new cluster is created.

  ⇒ The distances between this new cluster and the existing ones are not given. Some scheme has to be used to obtain these distances based on the two merged clusters. We call this the update of distances. Various schemes of updating distances will be described shortly.

- We can keep merging clusters until all the points are

merged into one cluster. A tree can be used to visualize the merging process. This tree is called a dendrogram.

- How do we get the distance between the new cluster and the existing clusters?

  $\Rightarrow$ The idea is to somehow aggregate the distances between the objects contained in the clusters.

  $\rightarrow$ For clusters containing only one data point, the between-cluster distance is the between-object distance.

  $\rightarrow$ For clusters containing multiple data points, the between-cluster distance is an agglomerative version of the between-object distances. There are a number of ways to accomplish this. Examples of these aggregated distances include the minimum , average or maximum between-object distances for pairs of objects across the two clusters.

  $\rightarrow$ Possibilities are: Single-link clustering (minimum distance), Complete-link clustering (maximum distance), Average linkage clustering, Centroid clustering and Ward's clustering.

- How do we decide which aggregation scheme to use?

⇒ Depending on how we update the distances, dramatically different results may come up.

⇒ Therefore, it is always good practice to look at the results using scatterplots or other visualization methods instead of blindly taking the output of any algorithm.

⇒ Clustering is inevitably subjective since there is no gold standard.

PCA

- Principal Component Analysis (PCA) is a method of dimension reduction.

- This is not directly related to prediction problems, but

  $\Rightarrow$ several regression methods are directly dependant on it, such as PCR and PLS, and

  $\Rightarrow$ it can be used as a part of EDA (exploratory data analysis)

- PCA uses the eigendecomposition of the covariance matrix, $\mathbf{\Sigma} = \mathbf{X}^T\mathbf{X}$, based on an SVD algorithm.

- Recall: The first principal component direction $\mathbf{v}_1$ has the following properties

  $\Rightarrow$ $\mathbf{v}_1$ is the eigenvector associated with the largest eigenvalue σ_1^2 of $\mathbf{X}^T\mathbf{X}$

  $\Rightarrow$ $\mathbf{z}_1 = \mathbf{X}\mathbf{v}_1$ has the largest sample variance amongst all normalized linear combinations of the columns of $\mathbf{X}$.

$\Rightarrow$ $\mathbf{z}_1$ is called the first principal component of $\mathbf{X}$ and we have $\mathrm{Var}(\mathbf{z}_1) = \sigma_1^2/N$. The second principal component direction $\mathbf{v}_2$ (the direction orthogonal to the first component that has the largest projected variance) is the eigenvector corresponding to the second largest eigenvalue, etc.

- The variance of the data along the principal component directions is associated with the magnitude of the eigenvalues.

DEEP Neural Networks

Overview of DNNs

- As seen above in the NN "zoo", there is an increasingly large number and diversity of neural networks

Definition 2. A deep neural network is a neural network with a level of complexity beyond that of a simple feed-forward network. Deep neural networks use sophisticated mathematical modeling to process data in complex ways. the adjective "deep" in deep learning refers to the use of multiple layers in the network. Methods used can be either supervised, semi-supervised or unsupervised.

- Here is an annotated list of some of the more common DNNs, with their application domains.

DNN Type	Applications
Convolutional-CNN	Image processing, object detection, natural language processing
Recurrent-RNN	Time series, natural language processing, speech recognition, machine translation
LSTM, GRU	Variants of RNN
Autoencoder	Noise filtering, Dimensionality reduction, image compression, anomaly detection
Adversarial-GAN	AlphaGo, Image generation, text generation, music generation
Graph-GNN	image classification, image recognition, object detection
Transformer-GPT	Natural language processing, machine translation

- **Multilayer perceptron (MLP)** is the simplest type of deep neural network. It is a feedforward network, which means that the data flows in one direction from the input layer to the output layer. MLPs are often used for classification tasks, such as image classification and natural language processing.

- **Convolutional neural network (CNN)** is a type of DNN that is specifically designed for processing images. CNNs use convolution operations to extract fea-

tures from images. They are often used for image classification, object detection, and natural language processing tasks that involve images.

- **Recurrent neural network (RNN)** is a type of DNN that is specifically designed for processing sequences of data. RNNs use recurrence connections to allow them to remember the past inputs. They are often used for natural language processing tasks, such as speech recognition and machine translation.

- **Long short-term memory network (LSTM)** is a type of RNN that is specifically designed to handle long-term dependencies. LSTMs use gates to control the flow of information through the network, which allows them to learn long-term patterns in data. They are often used for natural language processing tasks that require long-term memory, such as machine translation.

- **Autoencoder** is a type of neural network that can be used to compress data or to extract features from data. It consists of two parts, an encoder and a decoder. The encoder compresses the data into a latent representa-

tion, while the decoder reconstructs the data from the latent representation.

- **Generative adversarial network (GAN)** is a type of DNN that is used to generate new data. GANs consist of two networks, a generator and a discriminator. The generator is responsible for creating new data, while the discriminator is responsible for distinguishing between real and fake data. GANs are often used for image generation, text generation, and music generation.

- **Deep reinforcement learning (DRL)** is a type of machine learning that uses DNNs to learn how to behave in an environment. DRL agents learn by trial and error, and they are often used for robotics and game playing.

- **Graph Neural Networks (GNNs)** are a class of deep learning methods designed to perform inference on data described by graphs. GNNs are neural networks that can be directly applied to graphs, and provide an easy way to do node-level, edge-level, and graph-level

prediction tasks.

- **Transformer** is a type of DNN that is specifically designed for natural language processing tasks. Transformers use attention mechanisms to learn the long-range dependencies in text data. They are often used for machine translation and text summarization tasks.

Convolutional Neural Networks (CNN)

TBC (time permitting)

Recurrent Neural Networks (RNN)

TBC (time permitting)

References

1. C. M. Bishop, *Pattern Recognition and Machine Learning,* Springer, 2006. (downloadable)

2. G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning with Applications in R.* Springer. 2013. (downloadable)

3. K. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022. (`probml.ai`)

4. M. Asch. *Digital Twins: from Model-Based to Data-Driven.* SIAM, 2022. (extracts)