Statistics for Chemical Engineers:
From Data to Models to Decisions

Victor M. Zavala

Department of Chemical and Biological Engineering
University of Wisconsin-Madison

victor.zavala@wisc.edu

Chapter 6: Statistical Data Analysis and Learning (Part III)

Classification Models

- Statistical learning provides powerful tools to construct predictive models from data.
- Models explored here are generalizations of models that we have analyzed.
- We begin discussion with *classification* models.
- Have inputs $X = (X_1, ..., X_n)$ with obs x_ω that lie in continuous domain $\mathcal{D}_X \subseteq \mathbb{R}^n$.
- Have output Y with scalar obs y_ω that lie in (binary) domain $\mathcal{D}_Y = \{0, 1\}$.
- We postulate relationship between X and Y:

$$Y = g(X, \theta) + \epsilon$$

  where $\theta \in \mathbb{R}^n$ are params, $g(X, \theta)$ is model, and ϵ is a noise RV.
- Goal is to find params that best explains our obs (data):

$$y_\omega = g(x_\omega, \theta) + \epsilon_\omega, \ \omega \in \mathcal{S}$$

Classification Models

- Estimation problem is known as a *classification* problem.
- Inputs X are known as features or descriptors.
- Output Y is known as label or class.
- What is different about class models is binary nature of output Y.
- Class models have many apps and binary nature makes them intuitive to analyze.
- Think about, for example, a chemical classification problem: given features of chemical (e.g., its molecular structure), can we predict if this is toxic or not?
- Similarly, imagine you want to detect illness from blood pressure and glucose levels.

Classification Models

- To solve class problem, we use technique called logistic regression.
- Postulate model of form:

$$m(x,\theta) = \frac{1}{1+e^{-\theta^T x}}$$

- Mixture $\theta^T x = \theta_0 + \sum_{i=1}^n \theta_i x_i$ is known as *evidence*.
- Params attribute importance (weights) to different features.
- Postulated model is logistic function and captures binary 0-1 logic.
- Logistic function has this behavior:
  - $m(x,\theta) \to 1$ as $\theta^T x \to +\infty$ (large evidence)
  - $m(x,\theta) \to 0$ as $\theta^T x \to -\infty$ (small evidence)
  - $m(x,\theta) = \frac{1}{2}$ if $\theta^T x = 0$ (no evidence)
- Reversing sign of param has effect of reversing logic.
- Param θ_0 does not weight any input; known as bias (ignores evidence).

Classification Models

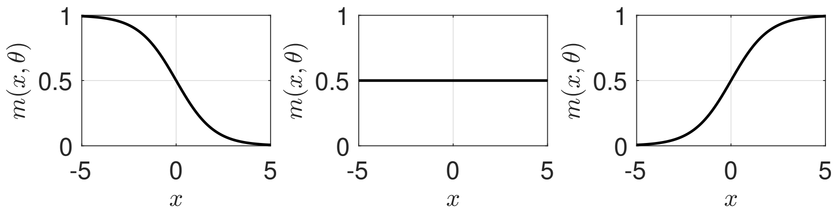


Figure: Behavior of logistic function for $\theta = -1$, $\theta = 0$, and $\theta = +1$.

Classification Models

- Function $m(x, \theta)$ is used to model conditional probs: $\mathbb{P}(Y = 1 \,|\, x, \theta)$ and $\mathbb{P}(Y = 0 \,|\, x, \theta) = 1 - \mathbb{P}(Y = 1 \,|\, x, \theta)$.
- Specifically, want to capture logic:
    - If evidence $\theta^T x$ is large, there is a high prob that $Y = 1$
    - If evidence $\theta^T x$ is small, there is a high prob that $Y = 0$
    - If no evidence ($\theta^T x = 0$), there is ambiguity (equally prob that $Y = 0$ or $Y = 1$)
- This is captured by defining conditional pdf:

$$f(y \,|x, \theta) = \mathbb{P}(Y = y|x, \theta) = m(x, \theta)^y (1 - m(x, \theta))^{1-y}.$$

- We apply MLE to estimate the parameters θ.
- Goal is to find $\hat{\theta}$ that max joint prob $\prod_{\omega \in \mathcal{S}} f(y_\omega \,|\, x_\omega, \theta)$:

$$\max_{\theta} \sum_{\omega \in \mathcal{S}} y_\omega \cdot \log(m(x_\omega, \theta)) + (1 - y_\omega) \cdot \log(1 - m(x_\omega, \theta))$$

Example: Classification for Fault Detection `ch6_gibbs_logistic.m`

- Imagine we have data $(P_\omega, C_\omega, T_\omega)$ for Gibbs reactor.
- Data is separated into two classes; one class represents normal mode (when T_ω is low) and we define corresponding labels as $y_\omega = 1$.
- Other class represent failure mode (when T_ω is high) and we define $y_\omega = 0$.
- Our goal is to develop a classification model that predicts mode of operation by using data on pressure and conversion $x_\omega = (P_\omega, C_\omega)$.
- In other words, model should be able to detect if reactor is failing by looking at a data pair of pressure and concentration. is that possible?

Example: Classification for Fault Detection `ch6_gibbs_logistic.m`

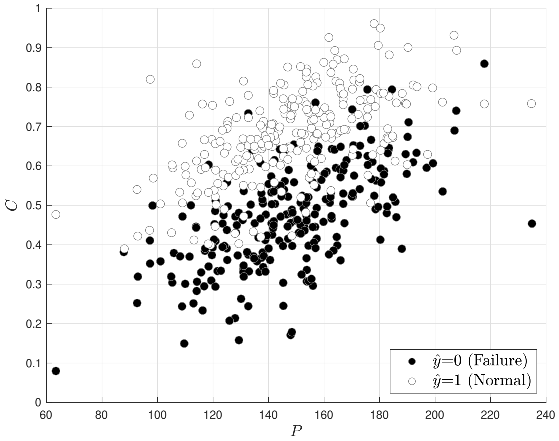


Figure: Gibbs reactor data showing separation between normal and failure operating modes.

Example: Classification for Fault Detection `ch6_gibbs_logistic.m`

- Model correctly predicts failures 85% of time (214 out of 250 obs).
- Logistic class also gives estimate of prob that predictions are in a given class.

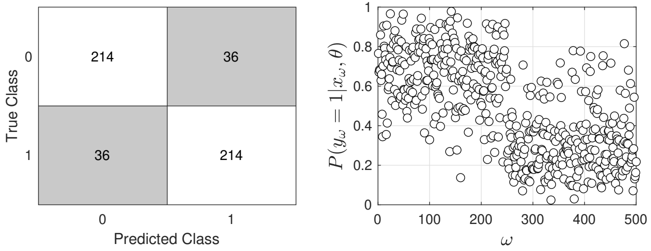


Figure: Confusion matrix (left) and classification probabilities (right) for Gibbs reactor data.

Kernel Models

- Kernel models are generalizations of predictive models that we have explored.
- These models have surprising properties:
  - Require handful of parameters to make predictions (regardless of number of inputs)
  - Capture complex nonlinearities (without need to postulate input-output model)
- Concepts have implications in techniques that we have discussed.
- For instance, kernel concepts can be used to capture nonlinearities in PCA.
- Key principle of kernel methods is that covariance of our data can be represented as a matrix function (rather than a fixed matrix).
- Such function (a.k.a. kernel function) captures complex nonlinear relationships present in our data.

Kernel Models

- To explain origin of kernel methods, we explore their use in regression.
- A flexible approach to construct models consists of *mixing* different model types.
- Assume that Y and $X = (X_1, ..., X_n)$ follow relationship:

$$Y = \sum_{j=1}^{m} \theta_j \phi_j(X) + \epsilon$$

  where $\phi_j(X),\ j = 1, ..., m$ is collection of basic models (known as basis functions).
- Here, output and input have continuous domains $\mathcal{D}_Y \in \mathbb{R}$ and $\mathcal{D}_X \in \mathbb{R}^n$.
- Predictive model is mixture $m(X, \theta) = \sum_{j=1}^{m} \theta_j \phi_j(X)$.
- Define basis vector $\phi(X) = (\phi_1(X), ..., \phi_m(X))$ and write $m(X, \theta) = \theta^T \phi(X)$.
- Basis $\phi(X)$ can be linear or nonlinear (e.g., polynomial, sigmoidal, log, exponential).
- Parameters $\theta \in \mathbb{R}^m$ are mixing coeffs for basis functions.

Kernel Models

- Determine $\hat{\theta}$ by solving:

$$\min_{\theta} \; \frac{1}{2} \sum_{\omega \in \mathcal{S}} (y_\omega - \theta^T \phi(x_\omega))^2 + \lambda \theta^T \theta$$

- This is Bayesian estimation with noise $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ and prior $\theta \sim \mathcal{N}(0, \lambda \mathbf{I})$.

- Express problem in compact form:

$$\min_{\theta} \; \frac{1}{2} (\mathbf{y} - \mathbf{\Phi}\theta)^T (\mathbf{y} - \mathbf{\Phi}\theta) + \lambda \theta^T \theta$$

  where $\mathbf{\Phi} \in \mathbb{R}^{S \times n}$ is data matrix with entries $\mathbf{\Phi}_{\omega,j} = \phi_j(x_\omega)$ and $\mathbf{y}$ is output vector.

- Data matrix $\mathbf{\Phi}$ is transformation of original input data.

- If basis functions are linear then $\mathbf{\Phi} = \mathbf{X}$ (get a standard linear estimation prob).

Kernel Models

- We define objective as $S(\theta)$; optimal estimate must satisfy $\nabla_\theta S(\theta)$:

$$\theta = \frac{1}{\lambda}\mathbf{\Phi}^T(\mathbf{y} - \mathbf{\Phi}\theta).$$

- Define residuals $\mathbf{r} = \frac{1}{\lambda}(\mathbf{y} - \mathbf{\Phi}\theta)$ and thus $\theta = \mathbf{\Phi}^T\mathbf{r}$.

- By substituting estimate in $S(\theta)$:

$$S(\theta) = \frac{1}{2}\mathbf{r}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{r} - \mathbf{r}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{r}^T\mathbf{\Phi}\mathbf{\Phi}^T\mathbf{r}.$$

- Data matrix $\mathbf{\Phi}$ always appear as $\mathbf{\Phi}\mathbf{\Phi}^T$; we define $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T \in \mathbb{R}^{S\times S}$ and thus:

$$S(\theta) = \frac{1}{2}\mathbf{r}^T\mathbf{K}\mathbf{K}\mathbf{r} - \mathbf{r}^T\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{r}^T K\mathbf{r}$$

- $S(\theta)$ can be defined in terms of $\mathbf{r}$ (params have been eliminated).

Kernel Models

- This suggests an alternative strategy to find $\hat{\theta}$:
  - Find optimal residual estimates $\hat{\mathbf{r}}$ by solving:
    $$\min_{\mathbf{r}} \frac{1}{2}\mathbf{r}^T\mathbf{K}\mathbf{K}\mathbf{r} - \mathbf{r}^T\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{r}^T\mathbf{K}\mathbf{r}$$
  - Recover $\hat{\theta} = \mathbf{\Phi}^T\hat{\mathbf{r}}$.
- It turns out, however, that $\hat{\theta}$ are *not needed at all* to make predictions.
- Specifically, optimal model prediction is:

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{\Phi}\hat{\theta} \\ &= \mathbf{\Phi}\mathbf{\Phi}^T\hat{\mathbf{r}} \\ &= \mathbf{K}(\mathbf{K} + \lambda I)^{-1}\mathbf{y}.\end{aligned}$$

- Consequently, prediction only depends on input data $\mathbf{K}$ and output data $\mathbf{y}$.
- Params $\hat{\theta}$ are *intermediary* variables.

Kernel Models

- Kernel matrix $\mathbf{K}$ captures interactions of input variables X.
- Specifically, given $x_\omega,\ \omega \in \mathcal{S}$, kernel matrix is:

$$\mathbf{K}_{i,j} = k(x_i, x_j),\ i, j \in \mathcal{S}$$

  where $k(x_i, x_j)$ is known as *kernel function*.

- Kernel generalizes strategies discussed; e.g., linear estimation corresponds to:

$$k(x_i, x_j) = x_i^T x_j,\ i, j \in \mathcal{S}$$

- Case of mixtures of basis functions corresponds to:

$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j),\ i, j \in \mathcal{S}$$

- Kernel function captures nonlinear interactions in input data.
- A widely used kernel is radial basis function (RBF):

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2),\ i, j \in \mathcal{S}$$

  where $\gamma \in \mathbb{R}_+$ is a param; this is also known as Gaussian kernel.

Kernel Models

- Gaussian kernel captures nonlinear interactions in input data.
- To see this, consider scalar x_ω and notice that:

$$\begin{aligned}
&\exp(-\gamma(x_i - x_j)^2) \\
&= \exp(-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2) \\
&= \exp(-\gamma x_i^2 - \gamma x_j^2)\left(1 + \frac{2\gamma x_i x_j}{1\,!} + \frac{(2\gamma x_i x_j)^2}{2\,!} + \dots\right) \\
&= \exp(-\gamma x_i^2)\exp(-\gamma x_j^2)\left(1 \cdot 1 + \left(x_i\sqrt{2\gamma/1\,!}\right) \cdot \left(x_j\sqrt{2\gamma/1\,!}\right)\right. \\
&\qquad \left. + \left(x_i^2\sqrt{(2\gamma)^2/2\,!}\right) \cdot \left(x_j^2\sqrt{(2\gamma)^2/2\,!}\right) + \dots\right)
\end{aligned}$$

- Gaussian kernel is a separable function and can be written as:

$$\exp(-\gamma(x_i - x_j)^2) = \phi(x_i)^T\phi(x_j)$$

- Where $\phi(x)$ is an *infinite* collection of polynomial basis functions:

$$\phi(x) = e^{-\gamma x}(1,\ x\sqrt{2\gamma/1\,!},\ x^2\sqrt{(2\gamma)^2/2\,!},\ x^3\sqrt{(2\gamma)^3/3\,!}, \dots).$$

- Here, a *single* param γ controls coefficients of basis functions.

Kernel Models

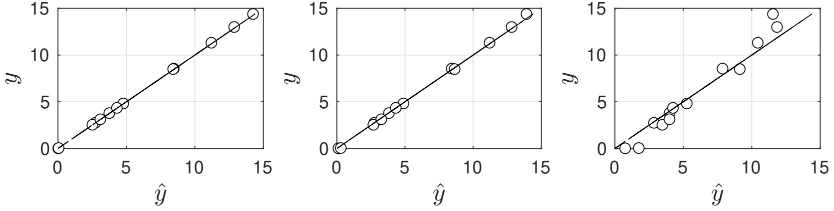
- In kernel methods, we do not need to postulate input-output model (e.g., $Y = \theta^T \phi(X)$) and estimate its params θ).
- Instead, we postulate kernel model and estimate its params γ.
- Kernel can be interpreted as a generating function of basis functions and requires few params (typically a handful).
- This is key advantage over standard estimation because overparam is avoided.
- Kernels can capture highly complex nonlinear behavior.

Example: Kernel Estimation for Hougen-Watson `ch6_hougen_watson_kernel.m`

- Consider Hougen-Watson model:

$$Y = \frac{(\theta_0 X_2 - X_3/\theta_4)}{(1 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3)}$$

- This model might not fit data if mechanism proposed is not correct.
- Instead of using this model, we use data (x_ω, y_ω) to construct a kernel model.
- We use $k(x_\omega, x_{\omega'}) = \exp(-\gamma \|x_\omega - x_{\omega'}\|^2)$ with $\gamma = 0.1, 0.001, 0.0001$.



Kriging

- Kriging (a.k.a. Gaussian process modeling) is a kernel modeling technique.
- In kriging, we postulate model of form:

$$y_\omega = m(x_\omega) + \epsilon_\omega,\ \omega \in \mathcal{S}$$

- In vector form:

$$\mathbf{y} = \mathbf{m} + \epsilon$$

- $\mathbf{y} \in \mathbb{R}^S$ contains obs y_ω and $\mathbf{m} \in \mathbb{R}^S$ contains model obs $m(x_\omega)$.
- In kriging we treat $\mathbf{m}$ as a random *function* that has no params.
- Motivated by fact that kernel methods do not require params θ to make predictions.
- This is key difference over parametric models $\mathbf{m}(\theta, x_\omega)$.

Kriging

- Model $\mathbf{m}$ is assumed to be RV $\mathcal{N}(0, \mathbf{K}(\gamma))$.
- Here, $\mathbf{K}(\gamma) = \mathrm{Cov}(\mathbf{m}) \in \mathbb{R}^{S \times S}$ is model covariance and γ are its params.
- $\mathbf{K}(\gamma)$ as a matrix function that generates samples of model $\mathbf{m}$.
- In parametric models we generate alternatives by tuning parameters, kriging generates alternatives via sampling.

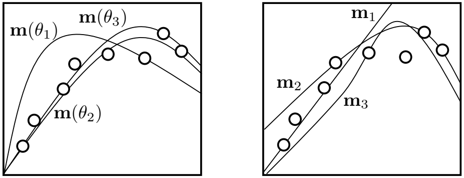


Figure: Generation of alternative models for parametric model $\mathbf{m}(\theta)$ using θ (left). Generation of alternative models for GP using realizations of $\mathbf{m} \sim \mathcal{N}(0, \mathbf{K}(\gamma))$ (right).

Kriging

- Covariance is a kernel matrix with entries:

$$\mathbf{K}_{\omega,\omega'}(\gamma) = k(x_\omega, x_{\omega'}, \gamma),\ \omega, \omega' \in \mathcal{S}$$

  where $k(x_\omega, x_{\omega'}, \gamma)$ is kernel function.

- Joint pdf of $\mathbf{y}$ and $\mathbf{m}$ is given by:

$$f(\mathbf{y}, \mathbf{m}|\gamma) = f(\mathbf{y}|\mathbf{m}, \gamma) f(\mathbf{m}|\gamma).$$

- Marginal pdf of $\mathbf{y}$ is:

$$f(\mathbf{y}|\gamma) = \int_{\mathbf{m} \in \mathcal{D}_{\mathbf{m}}} f(\mathbf{y}|\mathbf{m}, \gamma) f(\mathbf{m}|\gamma) d\mathbf{m}$$

Kriging

- One can show that marginal pdf of $\mathbf{y}$ is $\mathcal{N}(0, \mathbf{C}(\gamma))$.

- Entries of covariance matrix are:

$$\mathbf{C}_{\omega,\omega'}(\gamma) = \mathbf{K}_{\omega,\omega'}(\gamma) + \sigma^2 \cdot \delta_{i,j}, \quad \omega, \omega' \in \mathcal{S}.$$

  Here, $\delta_{\omega,\omega'} = 1$ if $x_\omega = x_{\omega'}$ and zero otherwise.

- One can estimate $\hat{\gamma}$ by using MLE with likelihood $f(\mathbf{y}|\gamma)$:

$$\max_{\gamma} \; -\frac{1}{2} \log |\mathbf{C}(\gamma)| - \frac{1}{2} \mathbf{y}^T \mathbf{C}(\gamma) \mathbf{y} - \frac{S}{2} \log 2\pi$$

- One needs to use advanced numerical solvers to tackle this problem.

Kriging

- Kriging makes predictions using conditional pdf $f(y|\mathbf{y}, \gamma)$.
- y is predicted output at new point x of interest.
- $\mathbf{y}$ are obs used to determine $\hat{\gamma}$ (obs used for training model).
- One can show that $f(y|\mathbf{y}, \gamma)$ is Gaussian $\mathcal{N}(m(x), \sigma^2(x))$ with:

$$m(x) = \mathbf{k}(x, \hat{\gamma})^T \mathbf{C}(\hat{\gamma})^{-1} \mathbf{y}$$
$$\sigma^2(x) = c(x, \hat{\gamma}) - \mathbf{k}(x, \hat{\gamma})^T \mathbf{C}(\hat{\gamma}) \mathbf{k}(x, \hat{\gamma})$$

where:

$$c(x, \hat{\gamma}) = k(x, x, \hat{\gamma})$$
$$\mathbf{k}_\omega(x, \hat{\gamma}) = k(x_\omega, x, \hat{\gamma}),\ \omega = 1, ..., S$$

- Mean prediction is $\mathbb{E}[y] = m(x)$ and variance is $\mathbb{V}[y] = \sigma^2(x)$.
- One use kriging to mix physics and data-driven models.
- Done by defining $f(\mathbf{y}|\mathbf{m}, \theta)$ as $\mathcal{N}(h(\mathbf{x}, \theta) + \mathbf{m}, \sigma \mathbf{I})$ where $h(\mathbf{x}, \theta)$ is physics model.

Example: Kriging Model for Gibbs Reactor `ch6_gibbs_kriging.m`

- Compare prediction of kriging (nonlinear) model against that of linear model.
- Gather data for inputs $X = (P, T)$ and output (conversion) $Y = C$.
- Kriging model better captures nonlinearity.

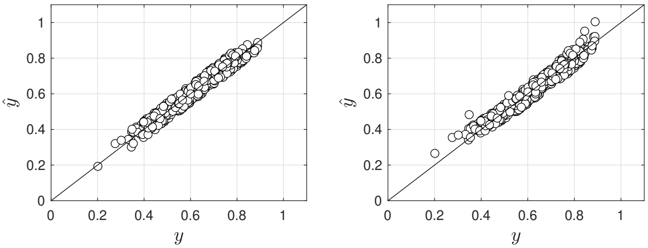


Figure: Parity plots for kriging (left) and linear (right) models for Gibbs reactor.

Example: Kriging Model for Gibbs Reactor `ch6_gibbs_kriging.m`

- Kriging provides uncertainty information for predictions (e.g., confidence intervals).

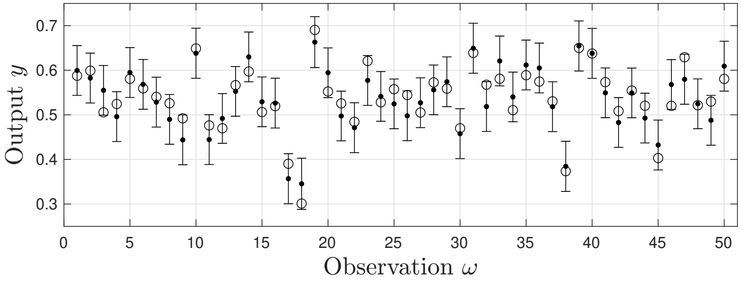


Figure: Observed outputs (white dots), mean outputs (black dots), and 95% confidence intervals.

Example: Experimental Design using GP Models `ch6_kriging_uq.m`

- In this example we illustrate how the uncertainty quantification capabilities of GP can be used to guide the collection of experiments.
- Imagine that you have a system in which you can manipulate an input x (e.g., temperature) to optimize the performance of an output y (e.g., cost).
- You start with 3 experimental data points that you use to train your GP. Note how the model prediction has no uncertainty at the location of the experimental points, but it has significant uncertainty in the unexplored regions of the input.
- Note how the uncertainty information provides guidance on where to sample next and provide an indication when the model has been approximated well.

Example: Experimental Design using GP Models `ch6_kriging_uq.m`

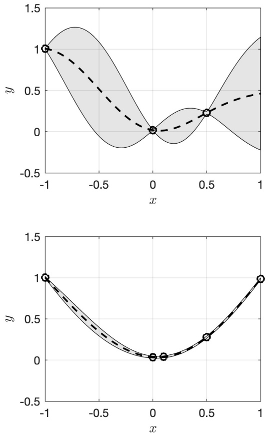


Figure: Evolution of model prediction uncertainty for GP with 3 data points (top-left), 4 data points (top-right), 5 data points (bottom-left) and 6 data points (bottom-right).

Neural Nets

- Explore powerful set of predictive models known as neural networks (NNs).
- Development motivated by following questions:
  - How do humans learn from observations?
  - How do humans establish connections between variables to make predictions?
- NNS are most general form of *parametric* models that currently exist.
- NNs mimic how brain (a network of neurons) learns from obs (data from senses).
- NNs are universal; they can capture virtually any relationship between variables.
- Flexibility helps explains why humans are remarkably good at learning.
- On the other hand, flex comes at expense of requiring an enormous number of params; one has to verify that NNs generalize well (predict outside domain of obs).

Fully Connected Neural Nets

- Fully connected NNs are parametric models of form:

$$Y = g(X, \theta) + \epsilon.$$

- Model $m(X, \theta)$ is built by using basis functions that are mixed *hierarchically*.
- Combinations of basis functions are captured by θ.
- Hierarchical structure mimics brain; the deeper the hierarchy, the more complexity.
- Goal is to train NN model to fit obss:

$$y_\omega = m(x_\omega, \theta) + \epsilon_\omega, \ \omega \in \mathcal{S}$$

- Determination of θ from data mimics human *learning* process.
- Specifically, NNs mimic how brain responds when exposed to data.

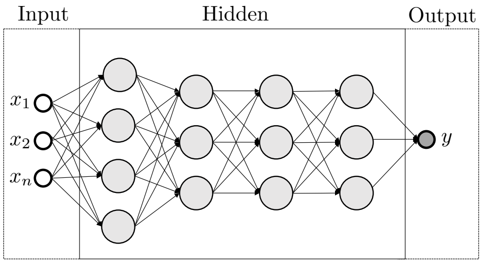
Fully Connected Neural Nets

- Neurons as processing units of brain that interact with one another through signals.
- Math description of a neuron is called a *perceptron*.
- Perceptron generates signal if evidence $\theta^T x = \sum_{i=1}^{n} \theta_i x_i + \theta_0$ is strong enough.
- Here, x_i are inputs with weights θ_i and param θ_0 is *bias*.
- Response of perceptron to evidence is modeled using binary logic:
$$a = \begin{cases} 1 & \theta^T x \geq \tau \\ 0 & \theta^T x < \tau \end{cases}$$
- Perceptron is unit that decides to activate or not when exposed to evidence.
- How to respond to different pieces of evidence is captured by params θ.
- Binary behavior is modeled using a logistic function:
$$a = \frac{1}{1 + e^{-\theta^T x}}.$$
- There are different types of activation functions.

Fully Connected Neural Nets

- NN mixes perceptron signals to construct *complex logic*.
- NN architecture consists of: input layer, hidden layers, and output layer:
    - Input layer contains perceptrons that take input data $x_\omega,\ \omega \in \mathcal{S}$ to generate signals. This captures basic logic (i.e., I predicted this because of that).
    - Hidden layers contain perceptrons that take signals from input layer to generate additional signals (captures complex logic).
    - Output layer contains perceptrons that take signals from hidden layer to generate final output signal $y_\omega,\ \omega \in \mathcal{S}$.
- Output signal can be continuous (regression) or discrete (classification).
- We now show how to train NNs; we consider architecture with one hidden layer.

Fully Connected Neural Nets



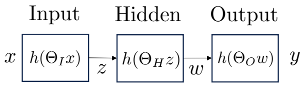


Figure: Sketch of hierarchical structure of a neural network (left). Simplified visualization showing sequential transformations of data (right).

Fully Connected Neural Nets

- *Input layer* is composed of $j = 1, ..., n_I$ perceptrons. Evidence in perceptron j is :
$$a_j = \sum_{i=1}^{n} \theta^I_{j,i} x_i + \theta^I_{j,0}$$
Given evidence, perceptron j generates the signal:
$$z_j = h(a_j)$$
where $h(a_j)$ is known as the activation function.

- *Hidden layer* is composed of $k = 1, ..., n_H$ perceptrons. Evidence in perceptron k is:
$$a_k = \sum_{j=1}^{n_I} \theta^H_{k,j} z_j + \theta^H_{k,0}.$$
Given evidence, perceptron k generates the signal:
$$w_k = h(a_k)$$

- *Output layer* takes signals of hidden layer as evidence:
$$a = \sum_{k=1}^{n_H} \theta^O_k w_k + \theta^O_0$$
Final output is $m = \sigma(a)$.

Fully Connected Neural Nets

- The propagation of the input x through the layers to obtain the output as a nested, *forward proagation* mapping:

$$\begin{aligned}\hat{y} &= h\left(\Theta_O h\left(\Theta_H h(\Theta_I x)\right)\right)\\ &= m(\theta, x),\end{aligned}$$

  where $\theta = (\Theta_I, \Theta_H, \Theta_O) \in \mathbb{R}^p$ contains all the parameters of the NN.

- Use MLE to estimate θ. For regression problems we solve:

$$\min_{\theta}\ L(\theta) = \frac{1}{2}\sum_{\omega\in\mathcal{S}}(y_\omega - m_\omega(\theta))^2 + \rho(\theta).$$

- For classification, one uses the log loss (as in logistic classification).

Fully Connected Neural Nets

- To solve problem, it is necessary to compute derivatives of loss function with respect to the parameters $\nabla_\theta L(\theta)$.

- These derivatives are computed by using a technique called *back propagation*.

- This technique marches in backward direction across NN (from output to input).

- Derivative information is used to progressively refine parameter estimates; typically using a family of algorithms known as gradient descent.

- In gradient descent algorithm, parameters are updated as:

$$\theta_{\ell+1} = \theta_\ell - \kappa \nabla_\theta L(\theta_\ell)$$

- Newton's method is not used in NNs (e.g., Hessian is difficult to compute).

Fully Connected Neural Nets

- NN params have no physical meaning (difficult to convey prior knowledge).
- Limitation manifests as need for large amounts of data to determine params which are also many (on the order of thousands to millions). In arch discussed, we have $n = n_I \cdot n + n_H \cdot n_I + n_H$ params.
- A typical issue with NNs is that of *overfitting*. Regularization (e.g., $\rho(\theta) = \lambda \theta^T \theta$) are often added to loss function.
- Uncertainty of estimated parameters increases with number of params.
- Because overfitting and high uncertainty are important issues, conducting cross-validation is critical.
- Hessian matrix $\nabla_\theta^2 L(\theta_\ell)$ is impossible to compute in many apps.
- One can mix physics and NNs by using models $Y = m(X, \theta) + NN(X, \theta)$.

Example: Neural Net Model for Gibbs Reactor `ch6_gibbs_neuralnet.m`

- We now compare performance of NN against kriging models.
- NN model is simple, contains one hidden layer with two perceptrons.
- NN fits data well; can capture high nonlinearity.
- Fact that this can be done with one hidden layer is remarkable.

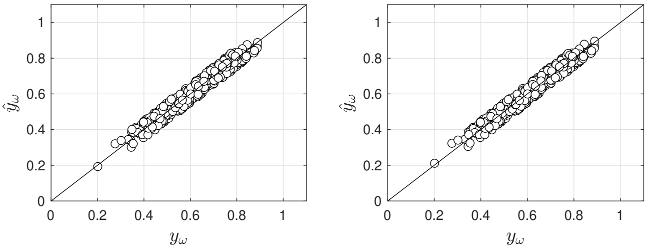


Figure: Fit of kriging (left) and NN (right) models for Gibbs model.

Example: Perceptrons as Basis Functions `ch6_neuralnet_basis.m`

- Why are NNs so effective at capturing such behavior?

- NNs are sophisticated basis function models (activation funcs are basis funcs).

- To see this, construct simple NN with N perceptrons:

$$h_j(a) = \frac{1}{1+e^{-a_j}},\ k = 1, ..., N$$

where $a_j = \theta^I_{j,1} \cdot x + \theta^I_{j,0}$ (NN only has one input).

- NN output is $\hat{y} = \sum_{j=1}^{n_I} \theta^O_j h(a_j)$ (there are no hidden layers).

- NN model has a total of 18 params.

Example: Perceptrons as Basis Functions `ch6_neuralnet_basis.m`

- Fitx of NN model to function $y = \sin(4x) + \sin(8x) + \exp(-x)$ is excellent.
- Here we show the sigmoidal functions (basis functions) of the NN model.
- Hessian reveals that several eigenvalues are close to zero.
- NN model is highly flexible but data is overfit.

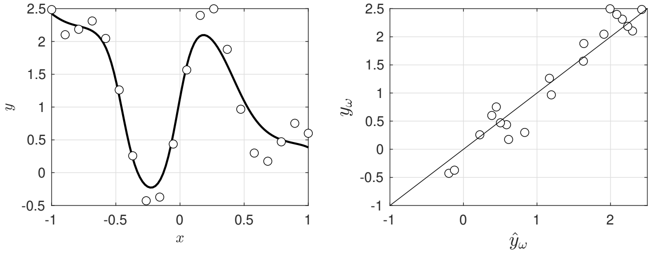


Figure: Fit of NN model to data (left) and parity plot (right).

Example: Perceptrons as Basis Functions `ch6_neuralnet_basis.m`

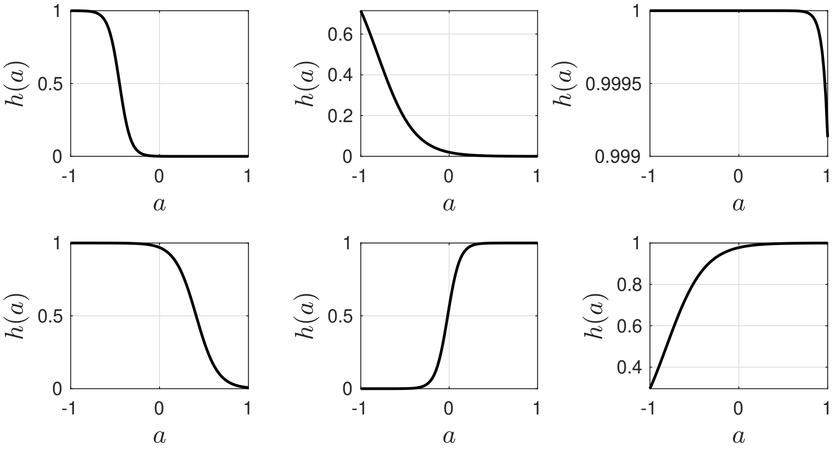


Figure: Sigmoidal functions of NN model for example model.

Convolutional Neural Nets

- CNNs are NNs that specialized to take structured data as inputs in form of vectors $\mathbf{x}_\omega$ (for 1D) or matrices $\mathbf{X}_\omega$ (for 2D) to make predictions y_ω.

- Input data vectors can be a sequence of numbers that encode a certain pattern and matrices can be images or other objects containing spatial patterns.

- e.g., can train a CNN to predict temp of a furnace from a picture of an infrared camera or to predict a property of a molecule from its sequence of atoms.

- CNNs can be generalized to process data that includes color images and can process 3D data in the form of tensors.

- CNNs extract knowledge from data in form of patterns (features).

- CNNs extract features by using filters (2D filter is matrix Θ_k that encodes pattern).

Convolutional Neural Nets

- Extracted features are used as evidence to generate activation signals.
- Process mimics how brain makes predictions based on patterns.
- Filter matrices and params of evidence are learned simultaneously from data.
- CNNs can learn what are best filters to extract from data to make predictions (not necessary for us to specify filters).
- Components of a CNN arch are: input layer, hidden layers, and fully connected layer.
- input and hidden layers seek to extract features from the input matrices.
- Features are passed to the fully connected layer to make a final prediction.

Convolutional Neural Nets

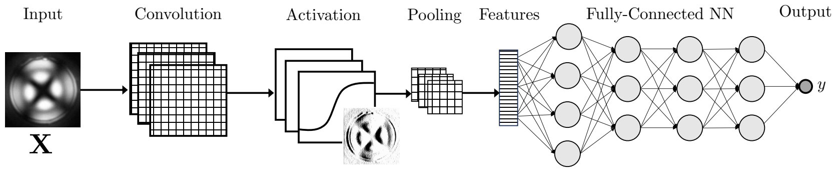


Figure: Sketch of a convolutional neural network architecture.

Convolutional Neural Nets

- *Input layer* is composed of $j = 1, ..., n_I$ perceptrons. Each unit performs convolution, activation, and pooling. Evidence in unit j is :

$$\mathbf{A}_j = \mathbf{X} * \Theta_j^I + \theta_j^I$$

  Here, Θ_j^I is param of perceptron (its filter). Matrix $\mathbf{X}$ is input data (e.g., image).

- Layer contains n_I filters. Filter matrices are small (3×3, 5×5 or 7×7).

- Convolution generates evidence $\mathbf{E}_j$ (matrix). Evidence is activated to produce signal:

$$\mathbf{Z}_j = h(\mathbf{A}_j)$$

  where $h(\mathbf{A}_j)$ is activation function; $\mathbf{Z}_j$ is activation matrix.

- Activation matrices $\mathbf{Z}_j$ are reduced in size by using pooling:

$$\bar{\mathbf{Z}}_j = p(\mathbf{Z}_j)$$

  where $p(\mathbf{Z}_j)$ is pooling function and $\bar{\mathbf{Z}}_j$ is pooled matrix.

Convolutional Neural Nets

- *Hidden layer* is composed of $k = 1, ..., n_H$ perceptrons (we use single layer).
- Evidence in perceptron k is:

$$\mathbf{A}_k = \sum_{j=1}^{n_I} \Theta_{k,j}^H * \bar{\mathbf{Z}}_j + \theta_k^H.$$

  Given evidence, signal generated is:

$$\mathbf{W}_k = h(\mathbf{A}_k)$$

  and this is reduced by using pooling as $\bar{\mathbf{W}}_k = p(\mathbf{W}_k)$.

- Matrices $\bar{\mathbf{W}}_k$ are converted to vectors $\mathbf{w}_k$ and stacked into feature vector $\mathbf{w} = (\mathbf{w}_1, ..., \mathbf{w}_{n_H})$.
- We summarize convolutions, activation, and pooling in input and hidden layers:

$$\mathbf{w} = f(\mathbf{X}, \Theta, \theta)$$

  Where Θ contains filters of all convolutional layers and θ contains all biases.

Convolutional Neural Nets

- *Fully connected layer* takes feature vector $\mathbf{w}$ and maps this to final output.
- NN contains it own inlet, hidden, and output layers.
- Mapping of the NN from the feature vector $\mathbf{w}$ to the output is summarized as:

  $$\hat{y} = NN(\mathbf{w}, \gamma)$$

  where γ are parameters of the NN input, hidden, and output layers.
- Given params for all layers $\beta = (\Theta, \theta, \gamma)$; CNN propagates input $\mathbf{X}_\omega$ to predict $\hat{y} = m_\omega(\beta)$.
- Use MLE to estimate β. For regression:

  $$\min_{\beta} \; L(\beta) = \frac{1}{2} \sum_{\omega \in \mathcal{S}} (y_\omega - m_\omega(\beta))^2 + \kappa\rho(\beta)$$
- A gradient descent algorithm uses derivative info to learn parameters.

Convolutional Neural Nets

- CNNs are conceptually similar to NNs; the key difference is that CNNs preprocess the input data to extract features prior to making a prediction.
- CNNs often have a large number of parameters; this is particularly evident from the fact that the parameters are matrices (the filter matrices).
- As such, CNNs are prone to *overfitting* and conducting cross-validation is also critical to make sure that the model generalizes well (to inputs not used for training).
- CNNs can take input objects that have multiple channels. For example, a color image contains red, blue, and green (RGB) channels.
- 1D CNNs can be used to process sequences (such as time series). 3D CNNs can process tensors (medical images and video).
- After training the CNN, one can analyze the filter matrices (Θ) to determine what patterns they are searching for in the images.
- Techniques such as Fourier transforms are used to accelerate computations. In many cases it is also necessary to use parallel computing to accelerate training process.

Example: Classification of Flow Cytometry `ch6_cnn_endotoxins_caseI.m`

- Endotoxins generated from bacteria can be detected by using solution that contains small droplets that are filled with a liquid crystal.
- Endotoxin binds to surface of droplet and generates perceptible change in configuration of liquid crystal inside droplets.
- Change in config can be measured using a flow cytometer; thousands of droplets are passed through cytometer and generates a light scatter field.
- Shape of scatter field changes with concentration and type of endotoxin.
- Train CNNs to automatically analyze scatter data to detect presence of endotoxin and to determine bacterial species that endotoxin is associated with.
- We train CNN to classify presence (or not) of endotoxin; to do so, we collect 98 observations (49 with no endotoxin present and 49 with endotoxin present).

Example: Classification of Flow Cytometry `ch6_cnn_endotoxins_caseI.m`

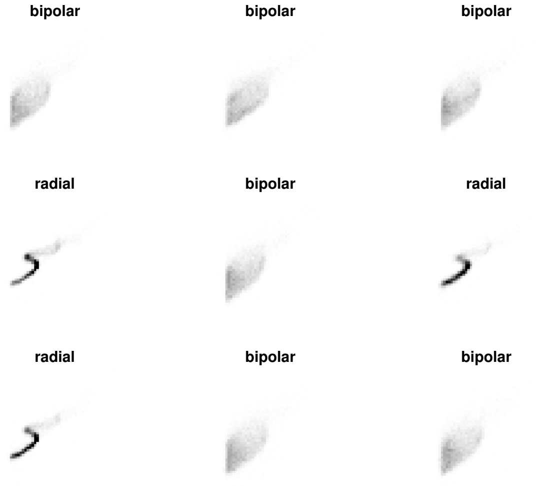


Figure: Example observations for light scatter fields for contaminated and clean solutions.

Example: Classification of Flow Cytometry `ch6_cnn_endotoxins_caseI.m`

- We train CNN with single hidden layer that contains 3x3 filter (this is simple CNN).
- We use an ReLU function to activate convolved images.
- Confusion matrix shows that CNN perfectly classifies scatter fields (100% accuracy).
- Spatial pattern of scatter fields contains significant information to tell them apart.

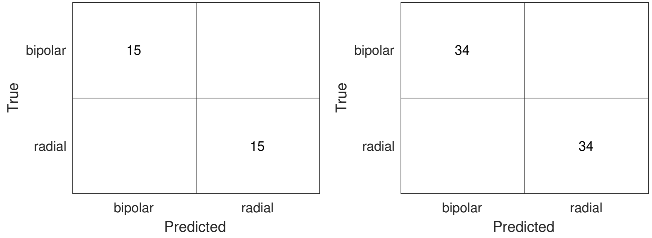


Figure: Confusion matrix for training set (left) and validation set (right).

Example: Classification of Flow Cytometry `ch6_cnn_endotoxins_caseI.m`

- Upon inspection, we find that filter matrix is of form:

$$\Theta = \begin{bmatrix} -75.0392 & -73.0502 & -74.4749 \\ -91.4429 & -79.6735 & -61.1360 \\ -89.0361 & -62.3274 & -38.6508 \end{bmatrix}$$

- Filter does not have a structure that we are familiar with (e.g., Sobel or Gaussian).
- To determine what filter is doing, we visualize activated images.
- CNN filter highlights shape of scatter field.

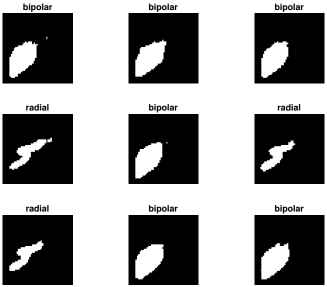


Figure: Activations of images shown in Figure 13 using CNN filter.

Example: Classification of Flow Cytometry `ch6_cnn_endotoxins_caseII.m`

- Now train CNN to distinguish bacterial species that generate endotoxin.
- Collect 84 obs; 24 for Escherichia Coli (EC), 28 for Pseudomonas Aeruginosa (PA), and 32 for Salmonella Minnesota (SM).
- We use 45 observations for training and 39 for validation.

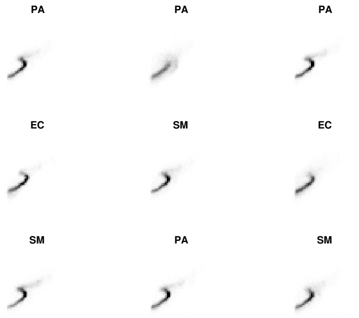


Figure: Example observations for light scatter fields three different bacterial species.

Example: Classification of Flow Cytometry `ch6_cnn_endotoxins_caseII.m`

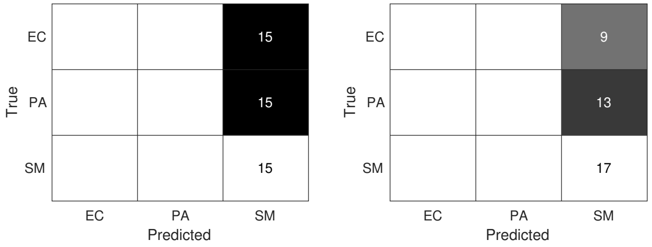


Figure: Confusion matrix for training set (left) and validation set (right) for classification of bacterial species.