Review DCOMP4702 - 2nd half

What if you only want to remember one thing...

To apply machine learning well, you need to understand how they work.

Good practices

- Implement (some of) the algorithms yourself
  - this forces you to pay close attention to details
- Stand on the shoulders of the giants
  - many excellent machine learning libraries (e.g. sklearn, PyTorch)
  - they can help you to be very productive
- Be creative
  - each algorithm has its own strengths and weaknesses
  - identify the needs/challenges of your applications, choose/mix-and-match/tweak algorithms to address them

What We Have Gone Through

How to build good ML models

- Making use of a crowd ⇒ Ensemble methods
  each of us is a biological prediction model trained on different datasets...
- Using a neural network ⇒ Neural networks
  brain-inspired models, some are good for images...
- Making a robust model ⇒ Robust machine learning
  malicious users, outliers,...
- Asking for explanations ⇒Interpretable machine learning
  ...let's ask the machines for explanations...
- Exploiting prior beliefs ⇒ Bayesian methods

This review lecture: We highlight some key ideas and important take-aways.

Ensemble Methods

- Ensembles can be much more expressive than basis models.
- Two approaches: independent methods, dependent methods
- Bagging: algorithm, model selection, why it works
- Random forest: algorithm, why it works, hyperparameters
- Boosting: weak learning and strong learning, AdaBoost
- Tuning ensemble methods: class of basis models, size of ensemble

Ensemble $>$ Basis Functions

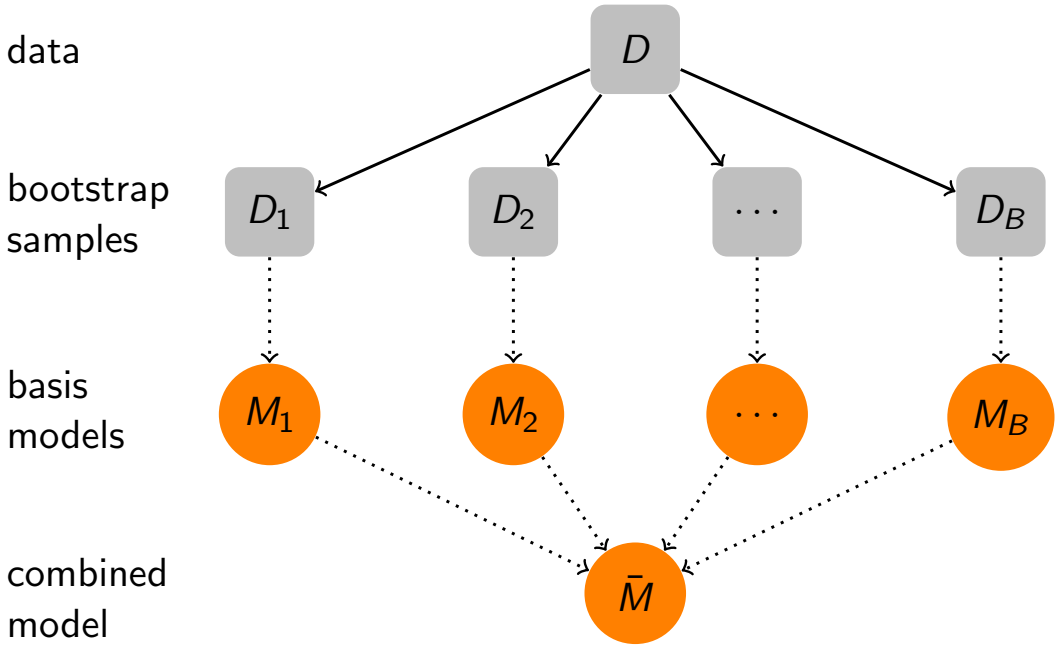
- An ensemble of basis models can often be used to represent a more complex functional relationship than each individual model can do.
- This is true even the basis models are "simple" (e.g. threshold classifiers or decision stumps).

  decision stump = one-level DT

Ensemble Learning

- Independent methods
  - Each model is trained independently of others.
  - e.g. bagging, random forest.
- Dependent methods
  - A model in the ensemble makes use of previously trained models.
  - e.g. AdaBoost.

Bagging



(*Classifcation*) $\bar{M}(x) = \text{majority}\{M_1(x), \ldots, M_B(x)\},$

(*Regression*) $\bar{M}(x) = \frac{1}{B}\sum_{i=1}^{B} M_i(x)$

Why bagging works

- The bias-variance decomposition
- Bagging reduces the variance while keeping the bias roughly the same

Random Forests

- Random forest is
  - a modified bagging method for DTs,
  - designed to further reduce variance by building a collection of *decorrelated* trees.
- RF and bagging with DTs differs *only* in how a decision tree is trained on a bootstrap sample
  - Bagging: for each node, choose the splitting variable from all d features.
  - RF: for each node, first randomly sample $m < d$ features, then choose the splitting variable among them.
- This subtle difference leads to further variance reduction.

Boosting

- Problem: if we have a weak learner, can we use it to build a strong learner?
- Surprisingly, the answer is yes, and an algorithm for converting a weak learner to a strong learner is called a boosting algorithm.
- Many boosting algorithms have been developed.

AdaBoost (Adaptive Boosting)

Problem

- Given: a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathcal{X} \times \{-1, +1\}$, and a set $G \subseteq \{-1, +1\}^{\mathcal{X}}$ of simple/weak/basis classifiers.
- AdaBoost produces a score model

$$F(x) = \sum_{t=1}^{T} \alpha_t f_t(x),$$

  where each α_t is non-negative and each $f_t \in G$.
- F classifies an instance x to the class $\text{sgn}(F(x))$.

AdaBoost

1: $F_0(x) = 0$.
2: Set $w_1(i) = \frac{1}{n}$ for each $i \in [n]$.
3: **for** $t = 1$ to T **do**
4: Train a classifier $f_t \in G$ on the weighted dataset $\{(w_i, x_i, y_i)\}$.
5: $\epsilon_t \leftarrow \sum_{i=1}^{n} w_t(i) I(f_t(x_i) \neq y_i)$.
6: $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$.
7: $F_t \leftarrow F_{t-1} + \alpha_t f_t$.
8: Set $w_{t+1}(i) \propto \exp(-y_i F_t(x_i))$ for each $i \in [n]$.
9: $F \leftarrow F_T / \sum_{t=1}^{T} \alpha_t$.

Essentially: sequentially reweigh examples and train new basis models.

Neural Networks

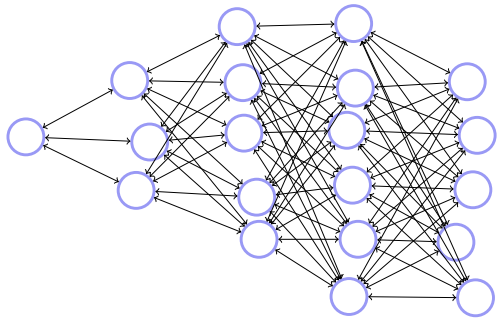
- Multilayer perceptrons (aka multilayer feedforward networks)
  - Specifying an MLP: structure and activation function
  - Forward propagation (compute output for a given input)
  - Backpropagation for gradient computation
- Gradient-based learning
  - Gradient descent and stochastic gradient descent
  - Backpropagation and automatic differentiation
- Implementing neural nets using PyTorch: `torch`, `torch.nn`, `torch.autograd`, `torch.optim`
- Motivations for deep networks: inspiration from nature, more compact representation, feature learning.

Convolutional neural nets

- They are special types of MLPs with sparse connections between layers.
- Three key architectural ideas: local receptive fields, weight sharing, sub-sampling.
- Two special types of layers
  - Convolutional layers
  - Sub-sampling layers

What are ANNs?

- An ANN is a network of basic computational units called (artificial) neurons

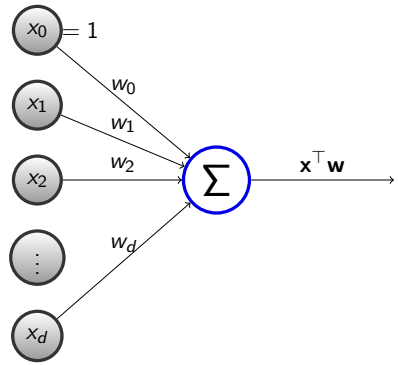


- The connection between two neurons may allow information to be sent in one direction or in both directions.
- Each neuron receives inputs along the incoming connections, and performs computes an output using simple transformations.
  - Typically, output is a simple transformation of a linear function of the inputs.
- We cover basic ANNs in this course.
  - More about ANNs: STAT3007/7007 Deep Learning

Simplest Neural Networks

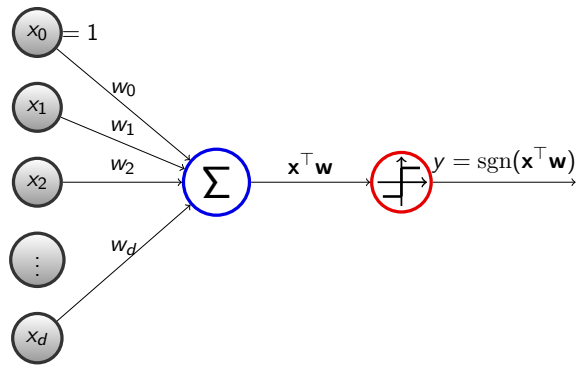
Linear regression

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$



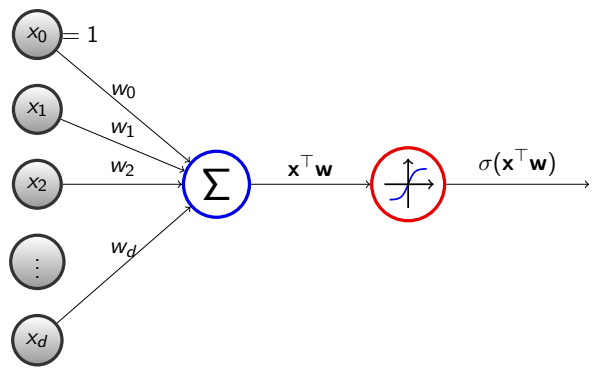
Perceptron

$$f(\mathbf{x}) = \begin{cases} +1, & \mathbf{w}^\top \mathbf{x} > 0, \\ -1, & \text{otherwise.} \end{cases}$$

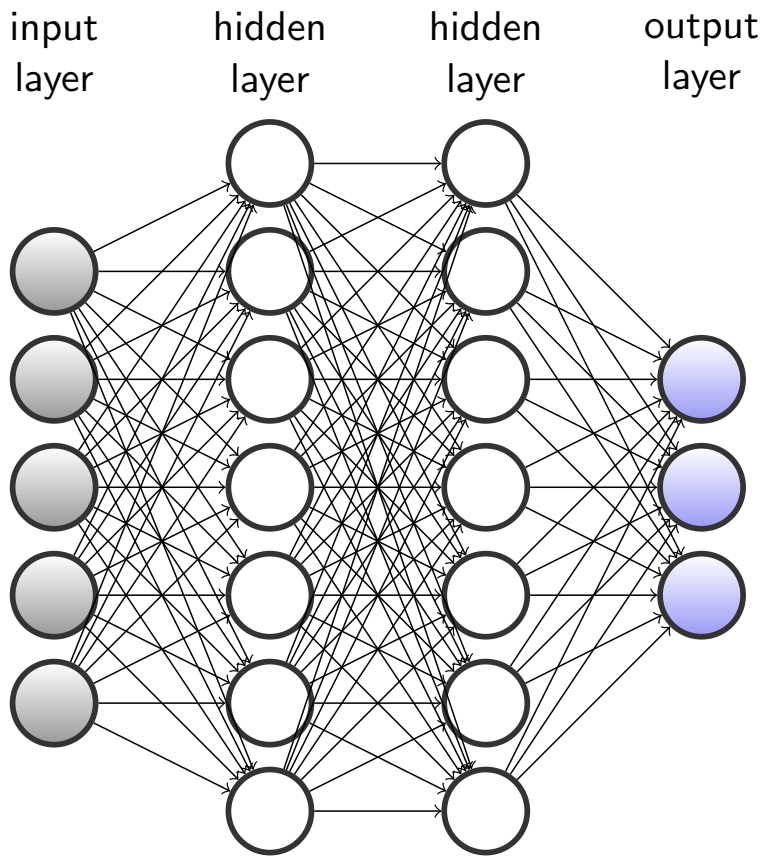


Binary logistic regression

$$f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$$



Multilayer Perceptron (MLP)



Regression Networks

- Consider a regression problem of predicting the value of an input $\mathbf{x} \in \mathbf{R}^d$.
- We usually design a neural net with a single output $o = f_{\mathbf{w}}(\mathbf{x})$, where $\mathbf{w}$ consists of all the network parameters.
- Given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbf{R}^d \times \mathbf{R}$, training/learning the neural net often amounts to minimizing the quadratic loss, or mean squared error (MSE)

$$\min_{\mathbf{w}} L(\mathbf{w}) \stackrel{def}{=} \frac{1}{n} \sum_{i=1}^{n} (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2.$$

Classification Networks

- Consider a classification problem of classifying an input $\mathbf{x} \in \mathbf{R}^d$ to one of C classes.
- We usually design a neural net $f_{\mathbf{w}} : \mathbf{R}^d \to \mathbf{R}^C$, with each output being the score for a class.
- The class with the largest score is the predicted class.
- Given a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbf{R}^d \times [C]$, learning the neural net often amounts to minimizing the log-loss

$$\min_{\mathbf{w}} L(\mathbf{w}) \stackrel{def}{=} \frac{1}{n} \sum_{i=1}^{n} -\ln p(y_i \mid \mathbf{x}_i, \mathbf{w}),$$

where $p(y_i \mid \mathbf{x}_i, \mathbf{w}) = e^{o_{i,y_i}} / \sum_{c=1}^{C} e^{o_{ic}}$, with $(o_{i1}, \ldots, o_{iC})$ being the network's output vector for $\mathbf{x}_i$.

Neural Network Learning

- Learning a neural net $f_{\mathbf{w}}(\mathbf{x}) \Rightarrow$ loss minimization

$$\min_{\mathbf{w}} L(\mathbf{w}) = \frac{1}{n} \sum_i L_i(\mathbf{w}),$$

  where $L_i(\mathbf{w})$ measures how the model $f_{\mathbf{w}}$ fits example i.
  - e.g., in regression , $L_i(\mathbf{w}) = (f_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2$.
- Gradient descent
  - choose $\mathbf{w}_0$
  - for $t \geq 0$, $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla L(\mathbf{w}_t)$

  where $\eta_t \geq 0$ is a step size (learning rate) to be chosen.
- Stochastic Gradient Descent (SGD)
  - choose $\mathbf{w}_0$
  - for $t \geq 0$, $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \tilde{g}_t$, where $\tilde{g}_t = \frac{1}{|S|} \sum_{i \in S} \nabla L_i(\mathbf{w})$, with S being a random subset of $1, \ldots, n$.

Backpropagation

- The backpropagation algorithm provides an efficient way to compute the gradient of the loss function of a feedforward neural net, which is essential in gradient-based learning.
- The algorithm performs a forward pass and a backward pass through the neural net
  - the forward pass propagates information from the input neurons to the output neurons to compute the outputs of all neurons
  - the backward pass propagates information from the output neurons to the input neurons to compute derivatives

Backpropagation: squared error, sigmoid network

1: Compute all o_i's.

2: For the output unit k,

$$\delta_k \leftarrow (o_k - y).$$

3: For each hidden unit i,

$$\delta_i \leftarrow o_i(1 - o_i) \sum_{j \in C(i)} w_{ij}\delta_j$$

when all input δ_j's have been computed.

4: For each connection (i, j),

$$g_{ij} \leftarrow \delta_j o_i.$$

Automatic Differentiation (Autodiff)

- Autodiff is a generalization of backpropagation
- Autodiff automatically compute the gradients for you
  - you only write function evaluation code, autodiff transforms it to gradient computation code
- Many machine learning software platforms now provide automatic differentiation (autodiff) tools

Neural Networks in PyTorch

- PyTorch provides several packages
  - `torch`: a general-purpose tensor package with GPU support
  - `torch.autograd`: a package for automatic differentiation
  - `torch.nn`: a neural net library with common layers and loss functions
  - `torch.optim`: contains common optimization algorithms
- We cover basics of these packages in this lecture.

OLS using PyTorch

```
import torch.optim as optim
import torch.nn as nn
from torch.nn.modules.loss import MSELoss

Y = Y.reshape(-1, 1)

net = nn.Linear(2, 1)
optimizer = optim.SGD(net.parameters(), lr=0.5, momentum=0)
mse = MSELoss()
for i in range(200):
    optimizer.zero_grad()
    loss = mse(net(X), Y)
    loss.backward()
    optimizer.step()

for param in net.parameters():
    print(param)
```

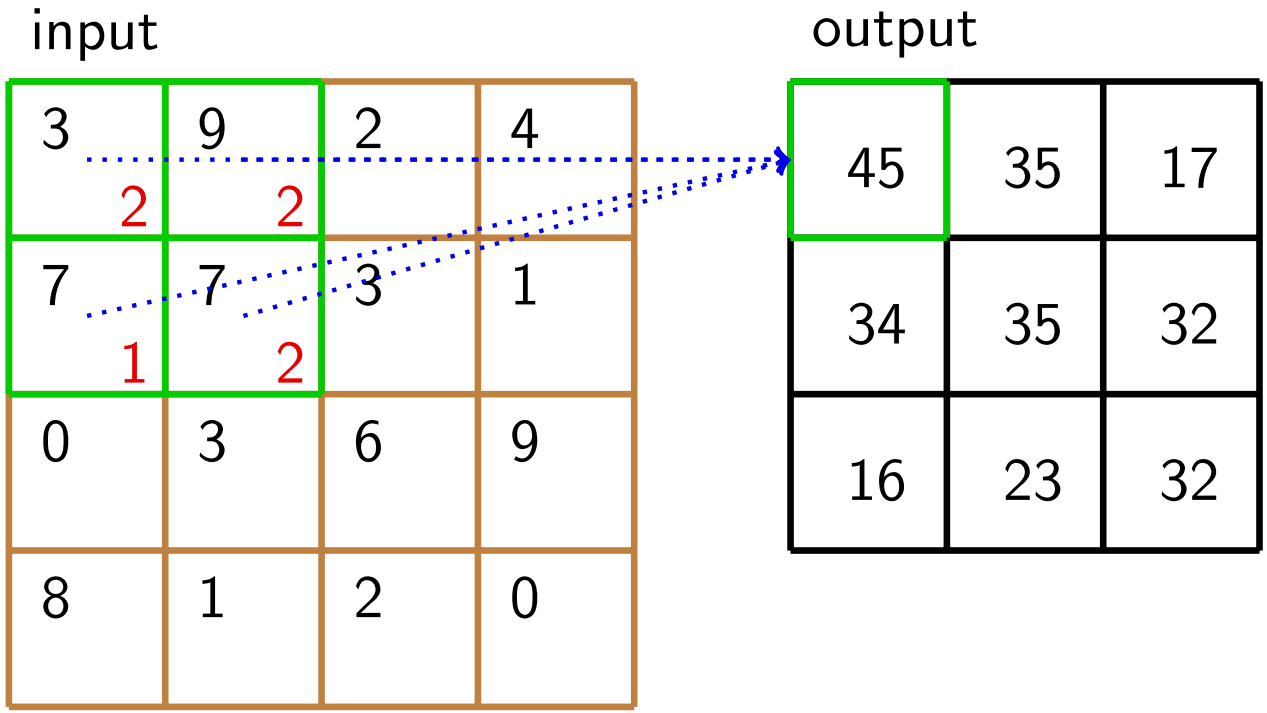
Convolutional Neural Nets (CNNs)

- CNNs are multilayer feedforward neural networks
  - they are MLPs where the weights have been constrained to mimic how biological vision works
- Three architectural ideas
  - Local receptive fields
  - Shared weights
  - Spatial or temporal sub-sampling

  These ensure some degree of shift, scale, and distortion invariance.

Convolution

- Convolution = sliding inner product

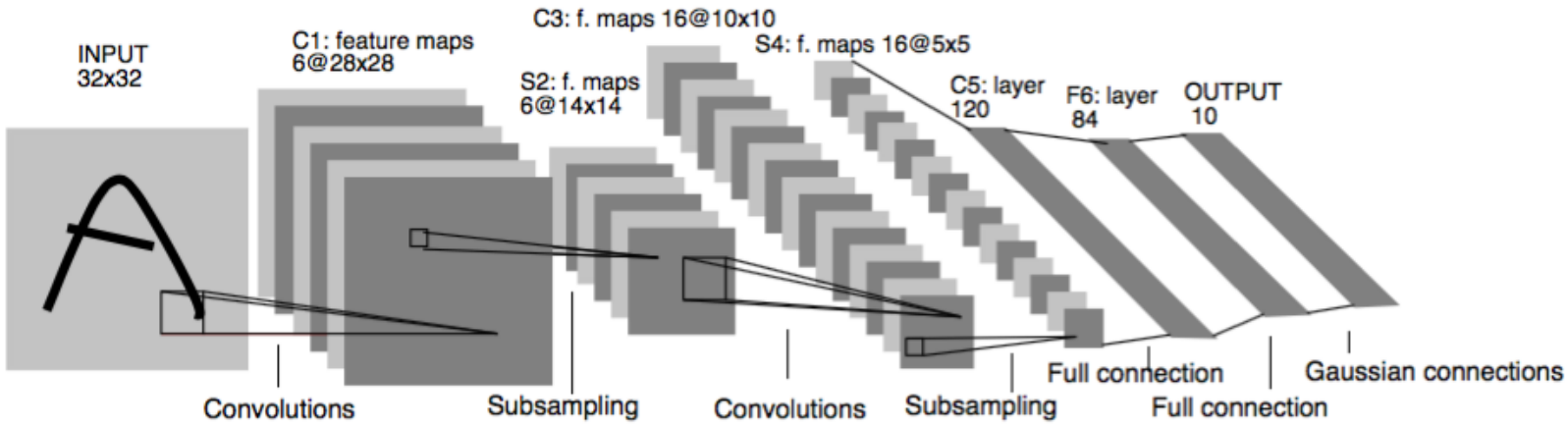


- Stride, zero-padding, dilation, and bias
- Convolution beyond 2D

Sub-sampling

- Sub-sampling (or pooling) is very similar to convolution.
- In average pooling, when we slide the filter through the input, we simply take the average of the input elements being scanned as the output.
- In max pooling, we replace average by max.
- The default stride is equal to the filter size (i.e. we do not pool the same element twice).

LeNet-5 (1998)

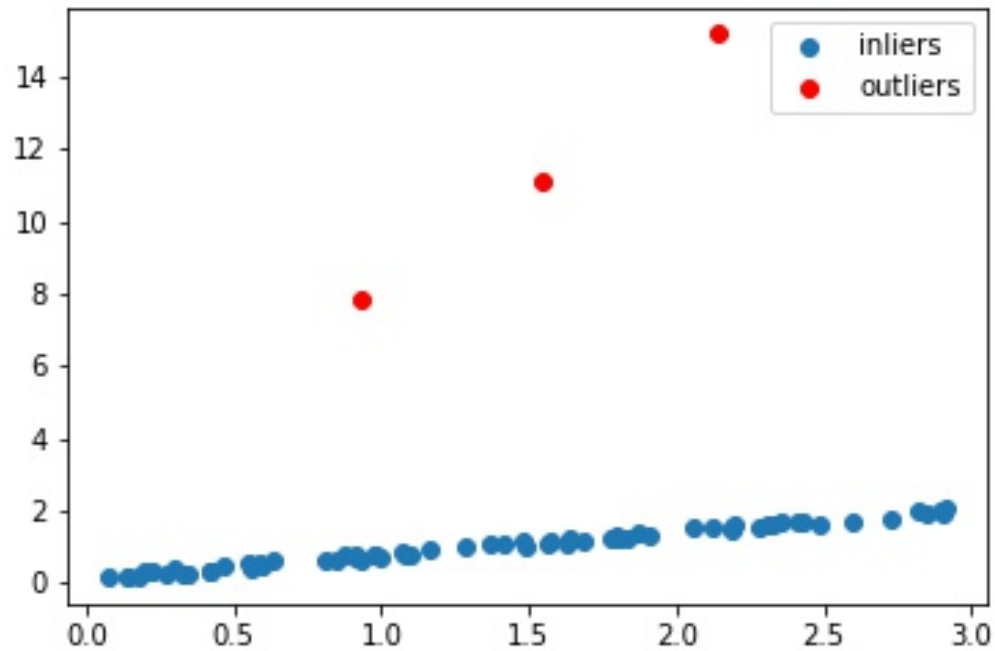

INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

Robust Machine Learning

- Robust machine learning methods try to produce models that work well with 'hard' data.
  - two types of hard data: outliers, adversarial examples
- Robust methods for outliers
  - Filtering before learning
  - Subsampling methods: Theil-Sen, RANSAC
  - M-estimators: LAD, Huber regression
- Robust methods for adversarial examples
  - Data augmentation approach
  - Adversarially robust learning objective

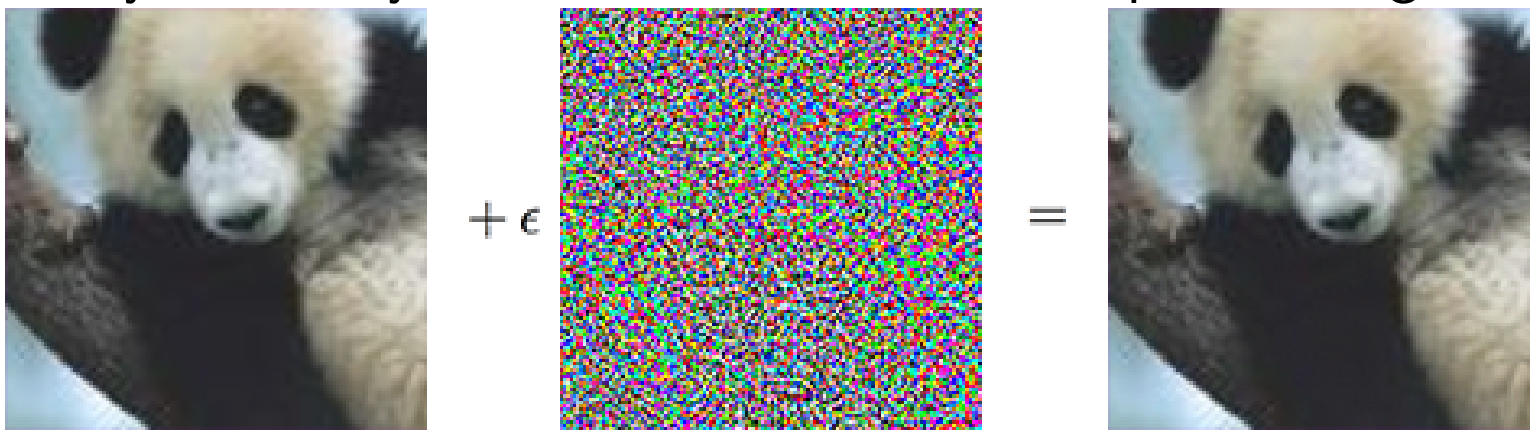
Outliers

- Outliers are unusual or atypical observations



Adversarial examples

- Can you see any difference between the two panda images?



- Adversarial examples are imperceptibly different from examples correctly classified by a model, but they are incorrectly classified.
- There are algorithms for generating adversarial examples
- An adversary can use adversarial examples to trick your system.

Goodfellow, Shlens, and Szegedy, Explaining and harnessing adversarial examples, 2015

Learning with Outliers

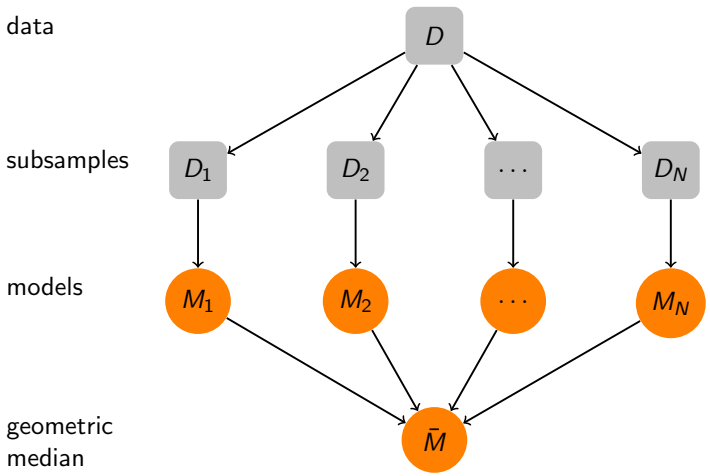
Approaches

- Filter outliers first, then build a model
- Subsampling methods
  - make use of multiple random subsamples to find a robust model
  - we cover Theil-Sen estimators and RANSAC
- Robust loss methods (aka M-estimators in statistics)
  - make use of a loss function which is robust against outliers
  - we cover ℓ_1 regression and Huber regression

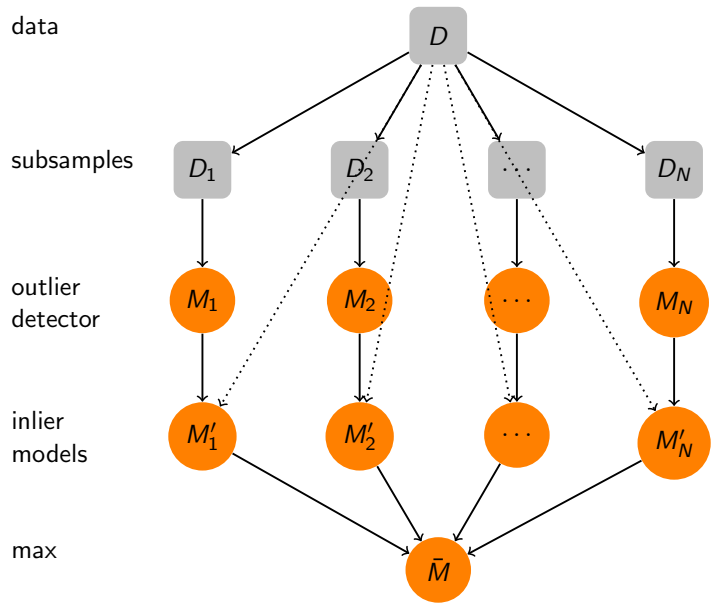
Subsampling Methods

Theil-Sen and RANSAC

Theil-Sen



RANSAC



Theil-Sen and RANSAC are implemented in `sklearn.linear_model` as `TheilSenRegressor` and `RANSACRegressor` respectively.

M-estimators

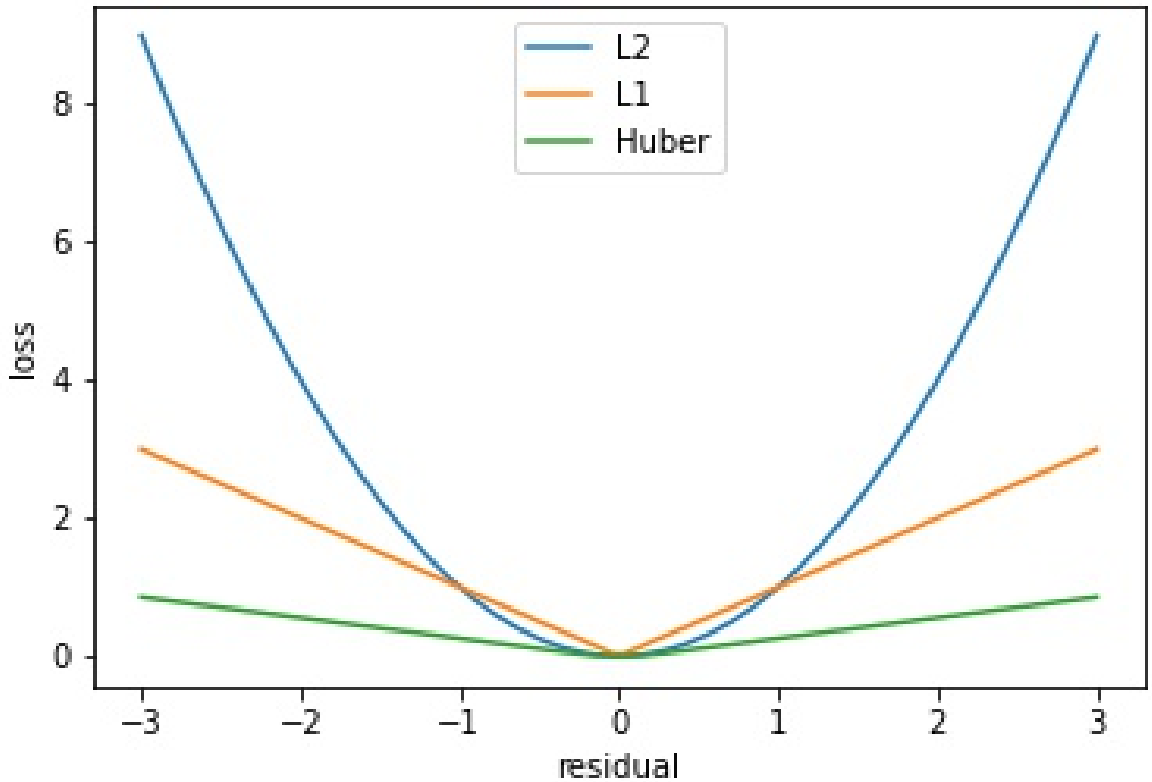
- Different losses

$$\text{(L2)} \qquad L_2(r) = r^2$$

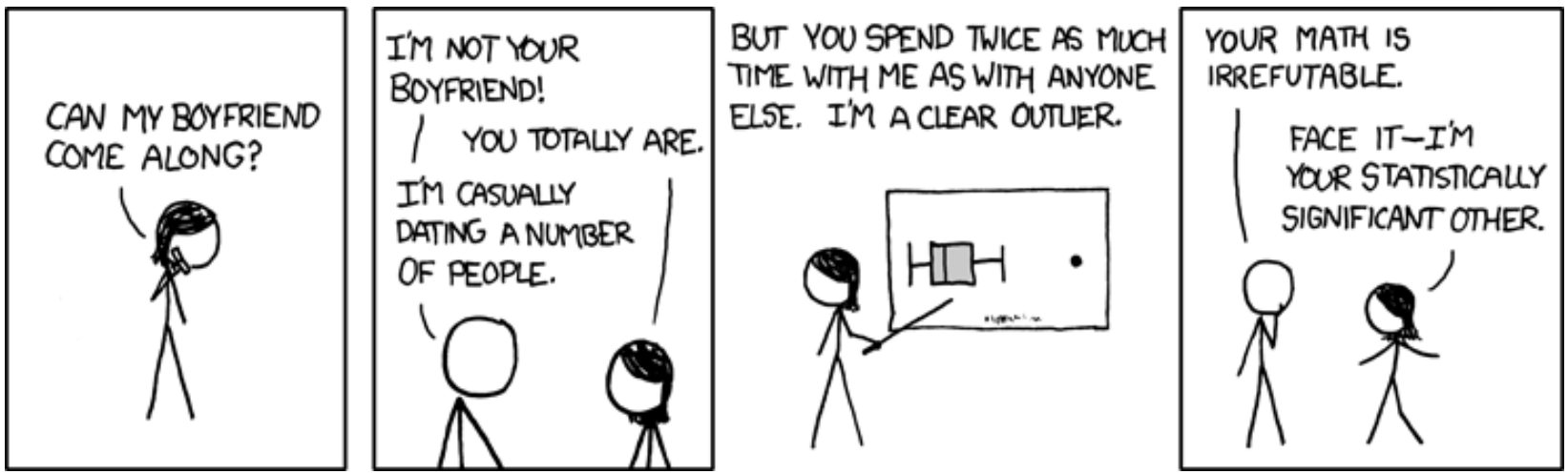
$$\text{(L1)} \qquad L_1(r) = |r|$$

$$\text{(Huber)} \qquad L_\delta(r) = \begin{cases} \frac{1}{2}r^2, & |r| \le \delta, \\ \delta\left(|r| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases}$$

- Different algorithms
  - OLS: minimizes L2 loss
  - LAD (least absolute deviations): minimizes L1 loss
  - Huber regression: minimizes Huber loss


L2
L1
Huber
loss
residual
8
6
4
2
0
−3
−2
−1
0
1
2
3

Outliers $\neq$ Liars, out



https://xkcd.com/539/

Adversarial Examples

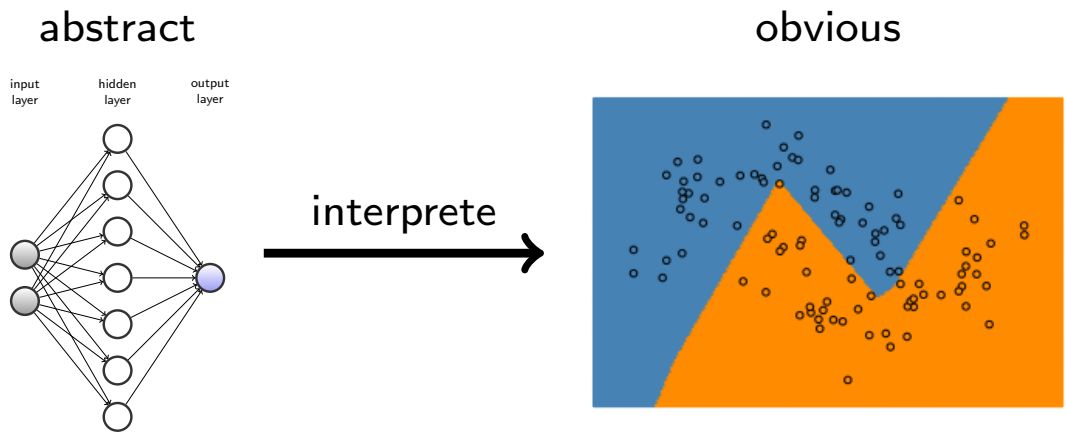
- Adversarial examples exist for all kinds of machine learning models.
- Defending against adversarial examples is hard.

Interpretable Machine Learning

- An interpretation
  - connects the abstract/unfamiliar to the obvious/familiar.
  - often tells part of the truth
- Approaches to make machine learning models interpretable
  - Built-in vs post hoc, white-box vs black-box, model-specific vs model-agnostic
- Some basic methods
  - interpretable models, surrogate method, variable importance, low dimensional approximation

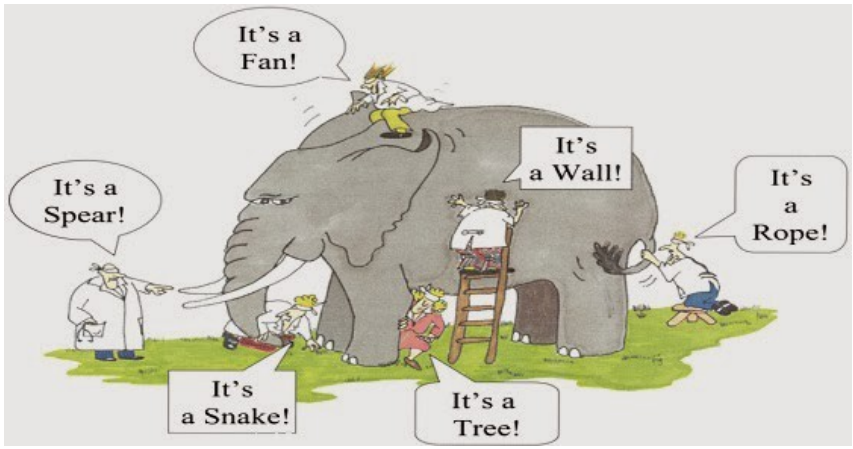
What is an interpretation?

- An interpretation connects the abstract/unfamiliar to the obvious/familiar.



Interpretation = Misinterpretation?

- Each interpretation often tells us part of the truth, and we may need to use several methods to form a more complete picture.



Approaches

- Various approaches have been taken to make machine learning models interpretable, and they can be categorized in various ways.
- Built-in vs post hoc
  - Built-in: models are designed to be interpretable (e.g. linear regression)
  - Post hoc: models are analyzed for interpretability (e.g. permutation importance)
- White-box vs black-box
  - White-box: everything about the model is needed (e.g. linear regression model weights)
  - Black-box: only partial information about the model is needed (e.g. permutation importance)

- Model specific vs model agnostic
  - Model-specific: designed for specific models only (e.g. linear regression model weights)
  - Model-agnostic: designed for generic learning approaches (e.g. permutation importance)

Some basic methods

- Interpretable models: linear regression, logistic regression, decision trees
- Surrogate model method
- Variable importance: Gini importance, permutation importance
- Low dimensional approximation

Bayesian Learning

- Bayesian learning overview
- Gaussian processes (GPs)
  - GPs as a generalization of multivariate Gaussians: mean function and kernel function
  - GPs as distributions on functions
  - computation of marginal distributions and conditional distributions
- GP regression
  - noisy-free and noisy observation models
  - prediction
  - model selection (maximum likelihood learning of hyperparameters)
- GP classification
- Implementing GPs in sklearn

Bayesian Learning Overview

Bayesian learning

- In Bayesian learning, we learn a distribution on all the models in Θ.
- Specifically, we assume a prior distribution $p(\theta)$ on Θ, and given a dataset D, we compute a posterior

$$\overbrace{p(\theta \mid D)}^{\text{posterior}} = p(\theta)p(D \mid \theta)/Z \propto \overbrace{p(\theta)}^{\text{prior}}\overbrace{p(D \mid \theta)}^{\text{likelihood}},$$

  where Z is the normalization constant.
- The posterior distribution $p(\theta \mid D)$ can be used in various ways when performing inference.

Inference problems

- Compute the MAP (maximum a posterior) model:

$$\theta_{\text{MAP}} = \operatorname*{argmax}_{\theta \in \Theta} p(\theta \mid D).$$

- Compute the (posterior) predictive distribution:

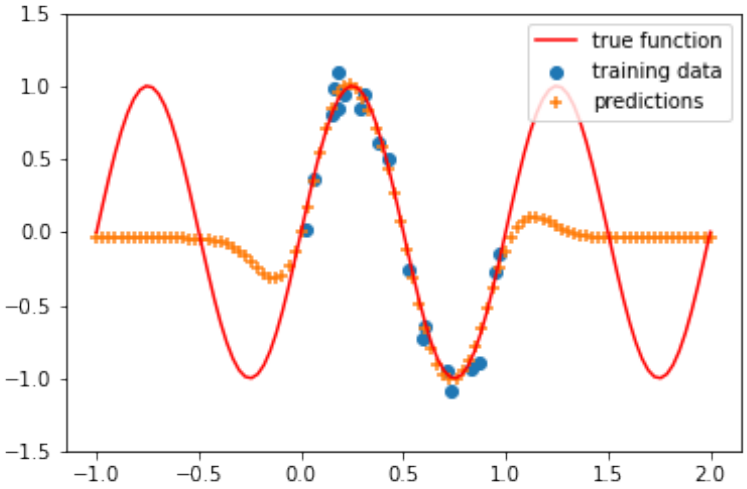
$$p(y \mid D, x) = \int p(y \mid \theta, x) p(\theta \mid D) d\theta.$$

- Compute posterior mean and variance of Y given x:

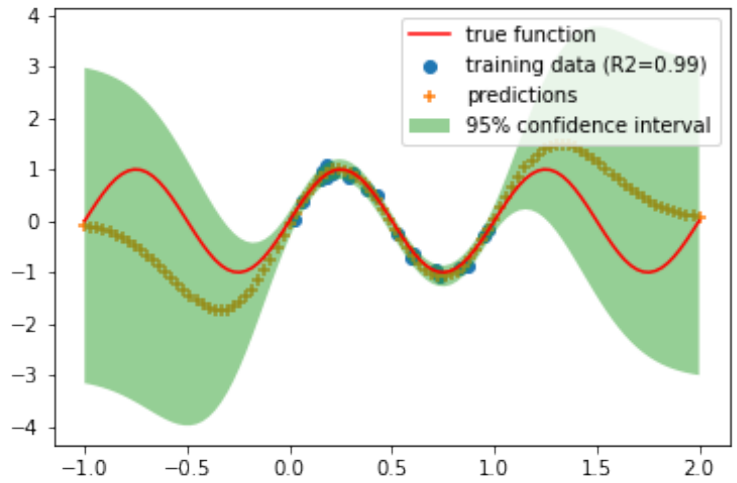
$$\text{posterior mean } \mu_x = \mathbb{E}(Y \mid x, D) = \int y p(y \mid D, x) dy$$

$$\text{posterior variance } \sigma_x^2 = \text{Var}(Y \mid x, D) = \int (y - \mu_x)^2 p(y \mid D, x) dy$$

From SVM to Gaussian Processes (GPs)



SVM (RBF kernel with $\gamma = 2000$)



Gaussian process (RBF kernel)

- SVMs and GPs both use regressors of the form

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}),$$

but GPs also provides a distribution on the output.

Gaussian Processes (GPs)

GPs generalize multivariate Gaussian distributions

- A GP is a collection of random variables such that any finite subset of which follows a (multivariate) Gaussian distribution.
- A GP can be specified in terms of the mean function m and the covariance function (aka kernel) k, defined by

$$m(Y) = \mathbb{E}(Y),$$
$$k(Y, Y') = \mathrm{cov}(Y, Y'),$$

where Y and Y' are any two random variables in the GP

GPs as distributions on functions

- Let the random variable $Y(\mathbf{x})$ denote the output for an input $\mathbf{x} \in \mathbf{R}^d$, and assume that $\{Y(\mathbf{x}) : \mathbf{x} \in \mathbf{R}^d\}$ is a GP.
- Define a random function F such that $F(\mathbf{x}) = Y(\mathbf{x})$, then the GP is a distribution for F:

$$F \sim GP(m, k),$$

where m and k are the mean function and the covariance function of the GP.

Posterior predictive distribution

- Observation noise $\epsilon \sim N(0, \sigma^2)$:

$$\boldsymbol{Y}' \mid \mathbf{X}', \mathbf{X}, \mathbf{y} \sim N\Big(\overbrace{\underset{t\times 1}{\mu_{\mathbf{X}'}} + \underset{t\times n}{K_{\mathbf{X}',\mathbf{X}}} \underset{n\times n}{(K_{\mathbf{X},\mathbf{X}} + \sigma^2 I)^{-1}} \underset{n\times 1}{(\mathbf{y} - \mu_{\mathbf{X}})}}^{\text{posterior mean}}, \tag{1}$$

$$\overbrace{\underset{t\times t}{K_{\mathbf{X}',\mathbf{X}'}} - \underset{t\times n}{K_{\mathbf{X}',\mathbf{X}}} \underset{n\times n}{(K_{\mathbf{X},\mathbf{X}} + \sigma^2 I)^{-1}} \underset{n\times t}{K_{\mathbf{X},\mathbf{X}'}}}^{\text{posterior covariance}}\Big). \tag{2}$$

- Model hyperparameters can be learned using maximum likelihood estimation.
- Commonly-used kernels: constant kernel, linear kernel, squared exponential kernel, Matérn kernel.
- Rules for constructing new kernels.

Final Remark

RF is an example of a tool that is useful in doing analyses of scientific data.

But the cleverest algorithms are no substitute for human intelligence and knowledge of the data in the problem.

Take the output of random forests not as absolute truth, but as smart computer generated guesses that may be helpful in leading to a deeper understanding of the problem.

Leo Breiman and Adele Cutler

Apply the same philosophy to other machine learning algorithms...

How to do this effectively:

Understand how the algorithms work, and follow good practices:

- Implement (some of) the algorithms yourself
- Stand on the shoulders of the giants
- Be creative