COMP0197
Applied Deep Learning

Andre Altmann

Department of Medical Physics and Biomedical Engineering

The UCL Hawkes Institute

a.altmann@ucl.ac.uk

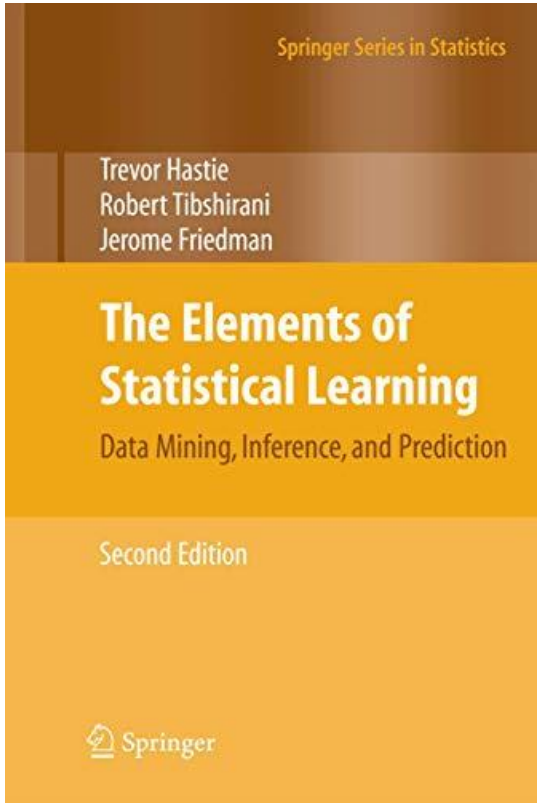
London, January 23rd 2025

Outline

- Deep Learning and Classic ML

- Sparse Data
  - Support Vector Machines
  - Kernels

- Ensembles
  - From Trees to Forests



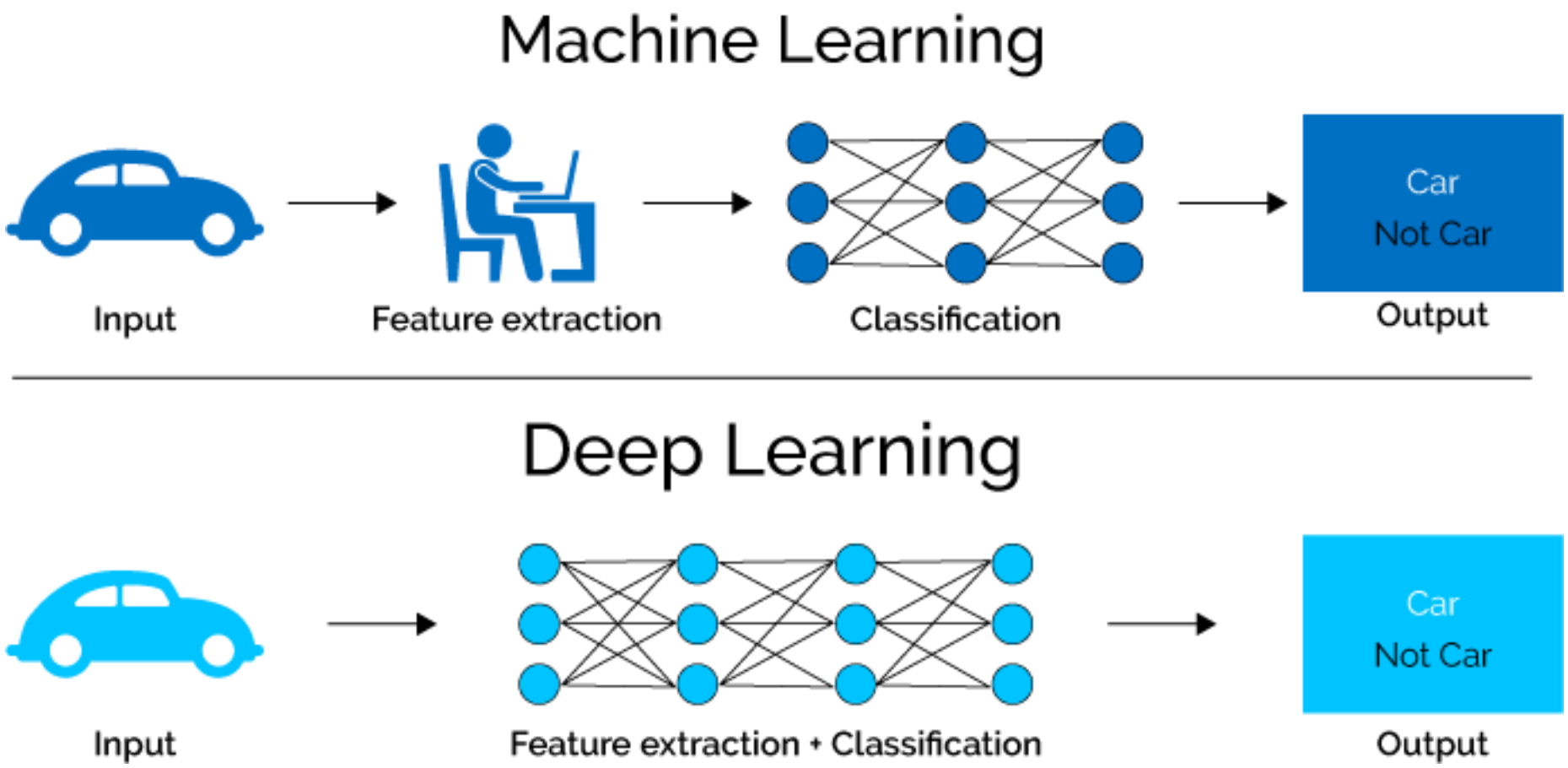
Deep And Classic

- Deep Learning is ‘just’ another ML method

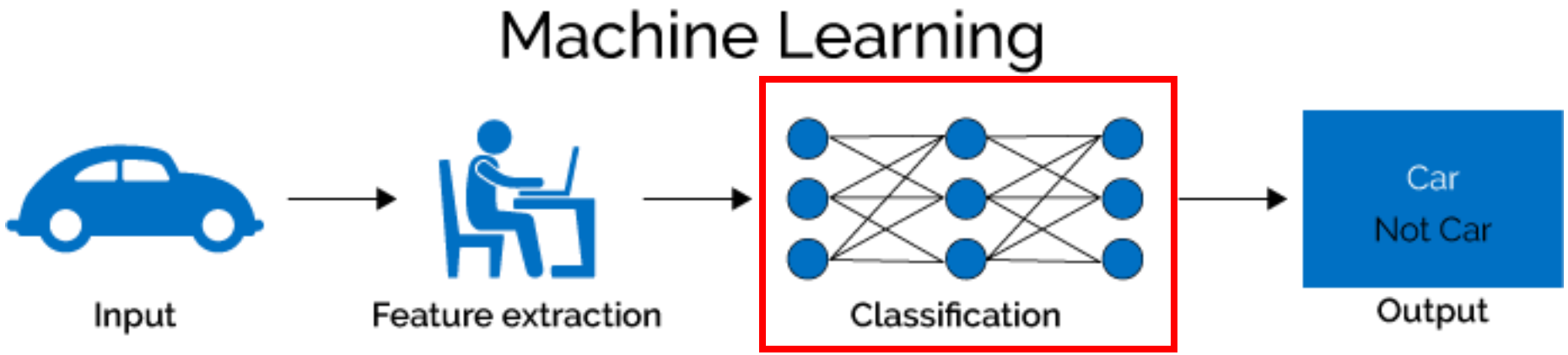
- DL is very powerful tool

- But: Not all problems match the requirements of DL
  1. Need Large Data
  2. Data Type
  3. Training Requires Substantial Resources
  4. Mostly a ‘Black Box’ model

Deep And Classic



Deep And Classic



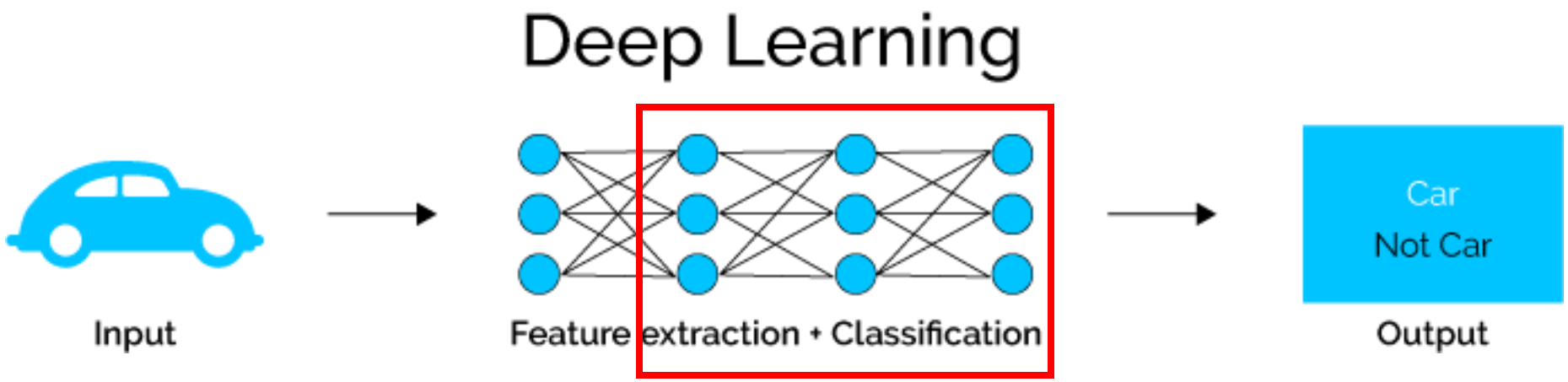
- Neural Nets can approximate many of the classic methods
- After all, machine learning can be viewed as optimizing:

$$\text{argmax}_\theta = \sum_{i=1}^{N} L(f_\theta(x_i), y_i) + \lambda\, \Omega(\theta)$$

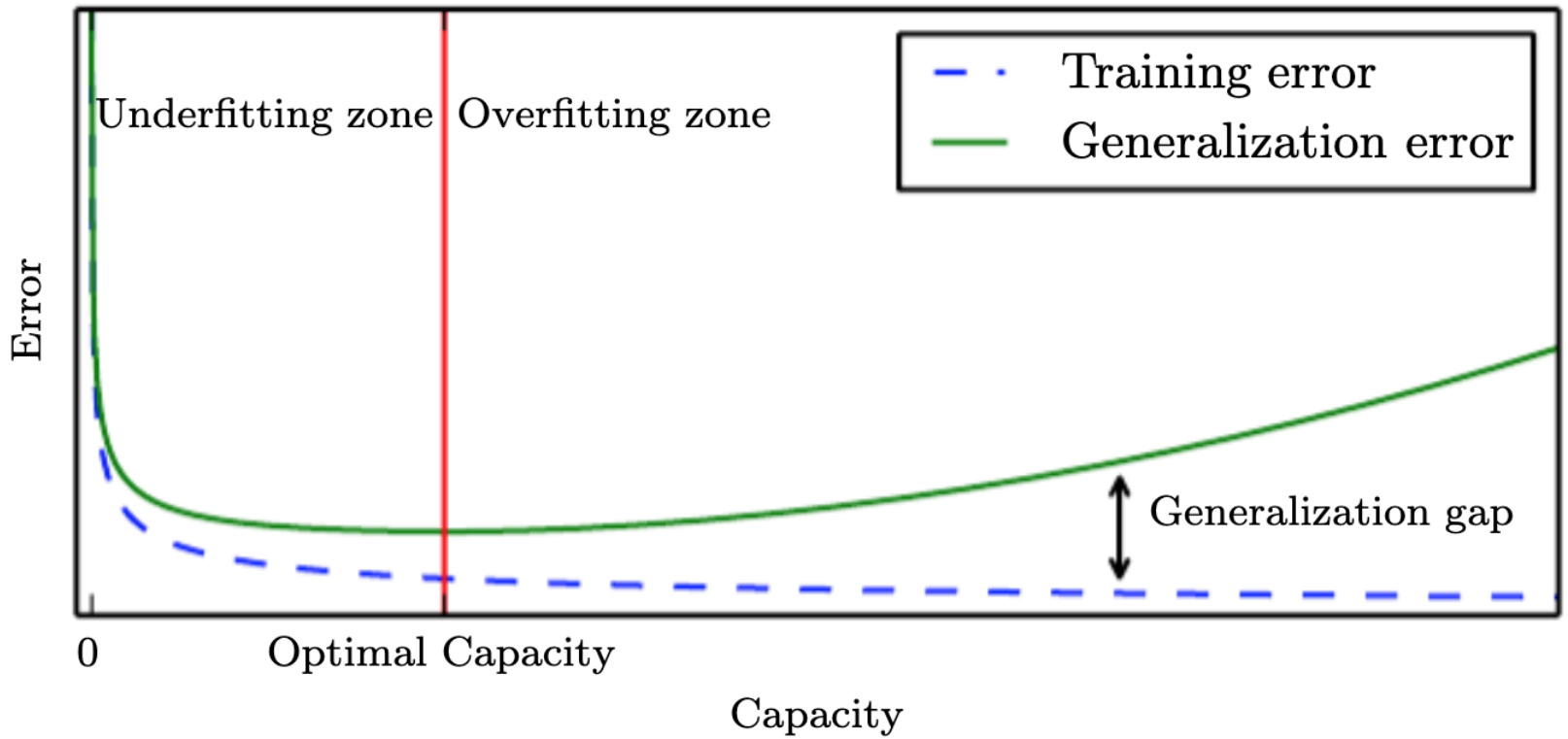
  - i.e., *f*, Loss, Regularization

Deep And Classic

- By setting f, L, Ω:
  - $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$
  - $\Omega(\mathbf{w}) = \|\boldsymbol{w}\|_2^2$
  - $L(\boldsymbol{w}) = \sum_{i=1}^{N} \{y_i \boldsymbol{w}^T x_i - \log(1 + \exp(\boldsymbol{w}^T x_i))\}$
- We get a Regularized Logistic Regression
- Changing Loss, Regularizer, and *f* gives us different ML methods



Double Descent



Further reading: https://arxiv.org/abs/1812.11118

Sparse Data

Sparse Data

Sparse Data

- Classic Machine Learning
  - Reduce feature dimensions
  - Use methods that work well on small data sets
    - e.g., SVM

- Learn features on other (unlabeled) data
  - Transfer-learning

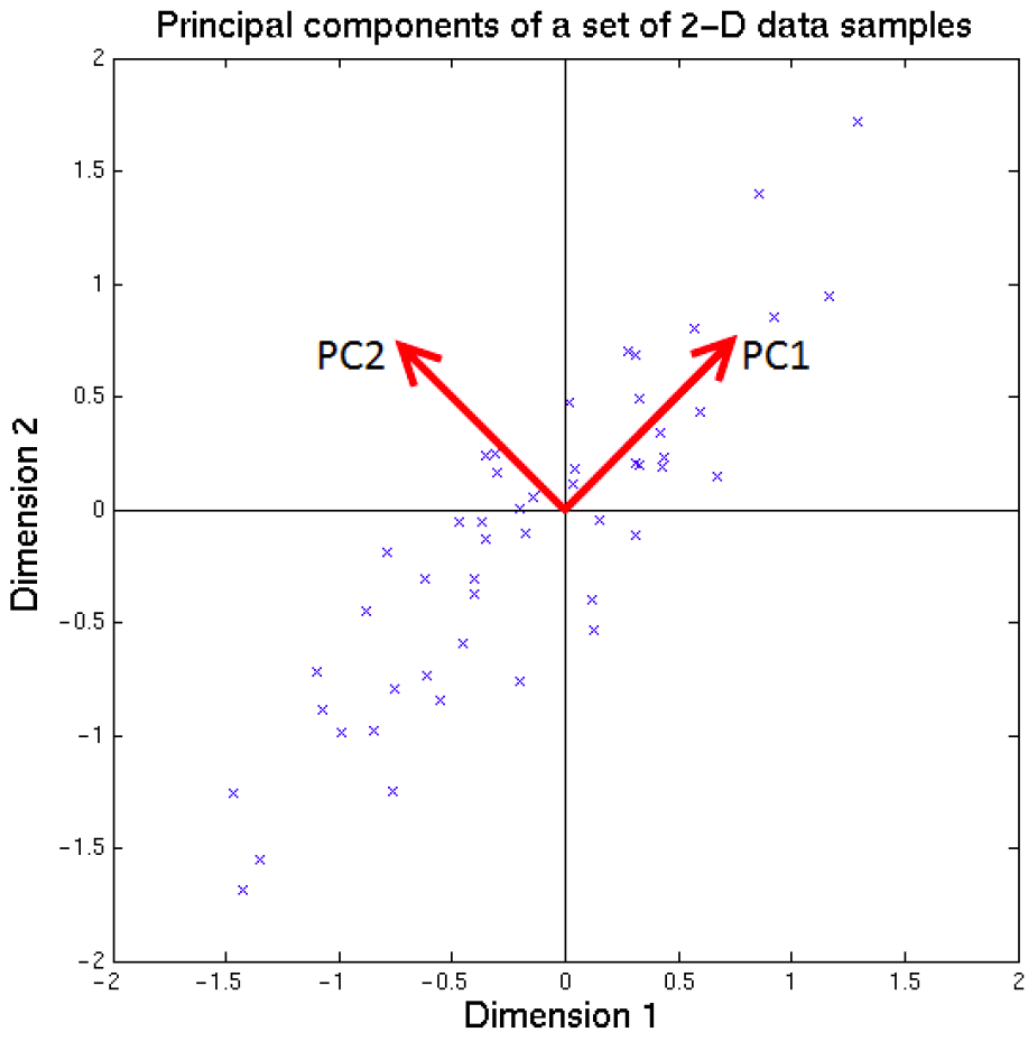
Reduce feature dimensions

Reduce Feature Dimensions

- Principal Component Analysis

- Consider a set of data $\{\boldsymbol{x}_i, i = 1, \ldots, N\}$, where $\boldsymbol{x}_i$ has some dimensionality d.

- The goal of PCA is to project the data onto a space having dimensionality $m < d$ while maximizing the variance of the projected data. We assume as before that m is given.



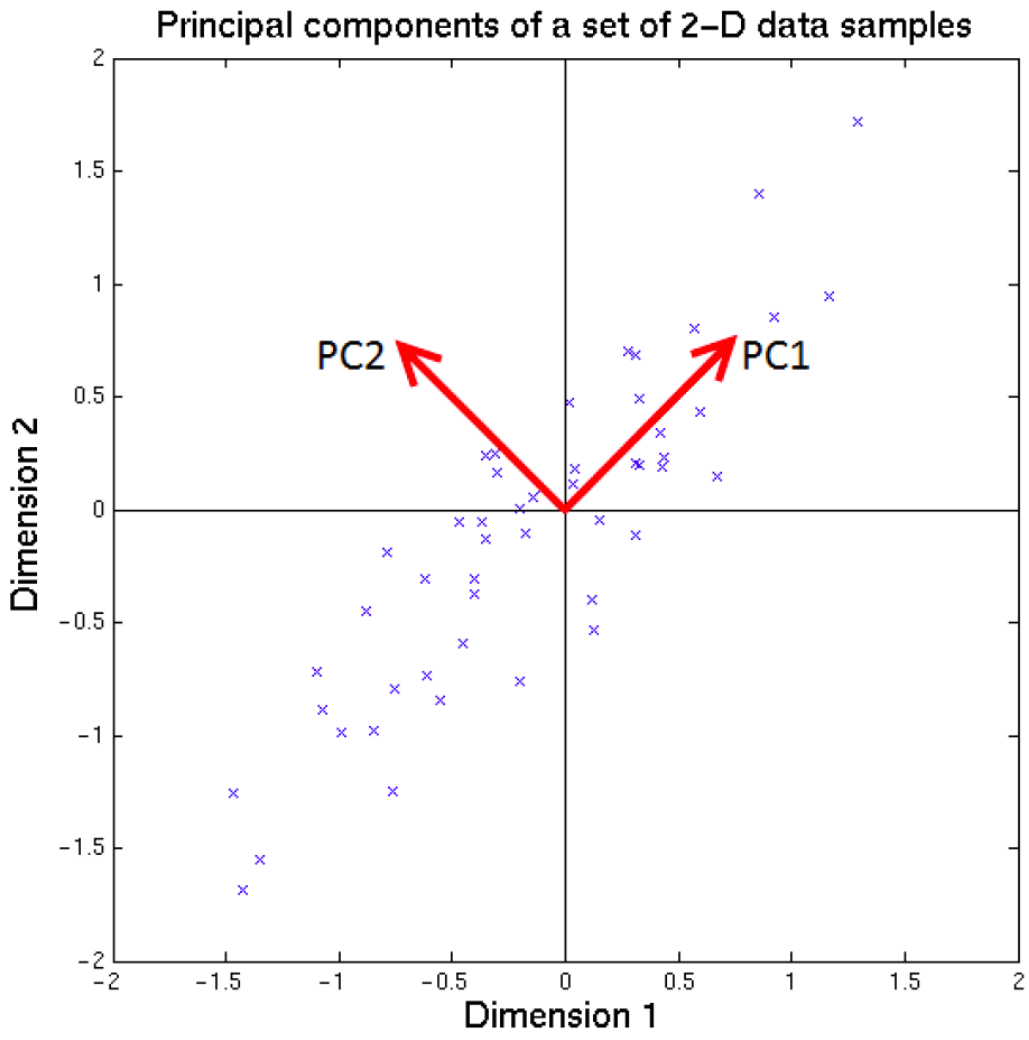
Reduce Feature Dimensions

- Principal Component Analysis

- The projection matrix composed of the eigenvectors corresponding to the m largest eigenvalues of the data covariance matrix is optimal

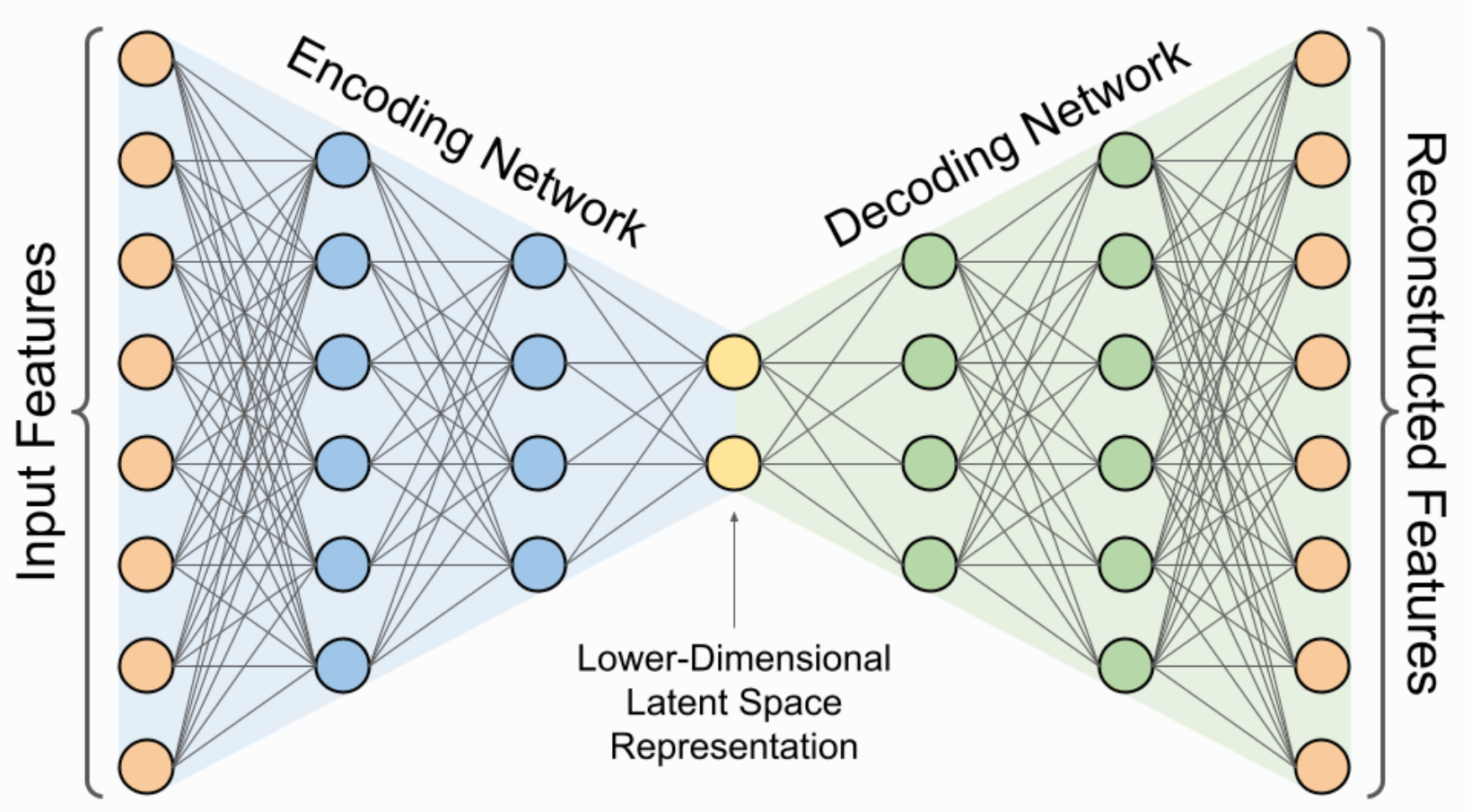
w, **V** = numpy.linalg.eig(numpy.cov(**X**))

- **w**: eigenvalues
- **V**: eigenvectors

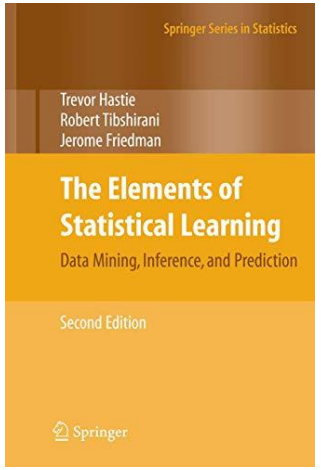


Reduce Feature Dimensions

- Deep Learning
  - Autoencoder

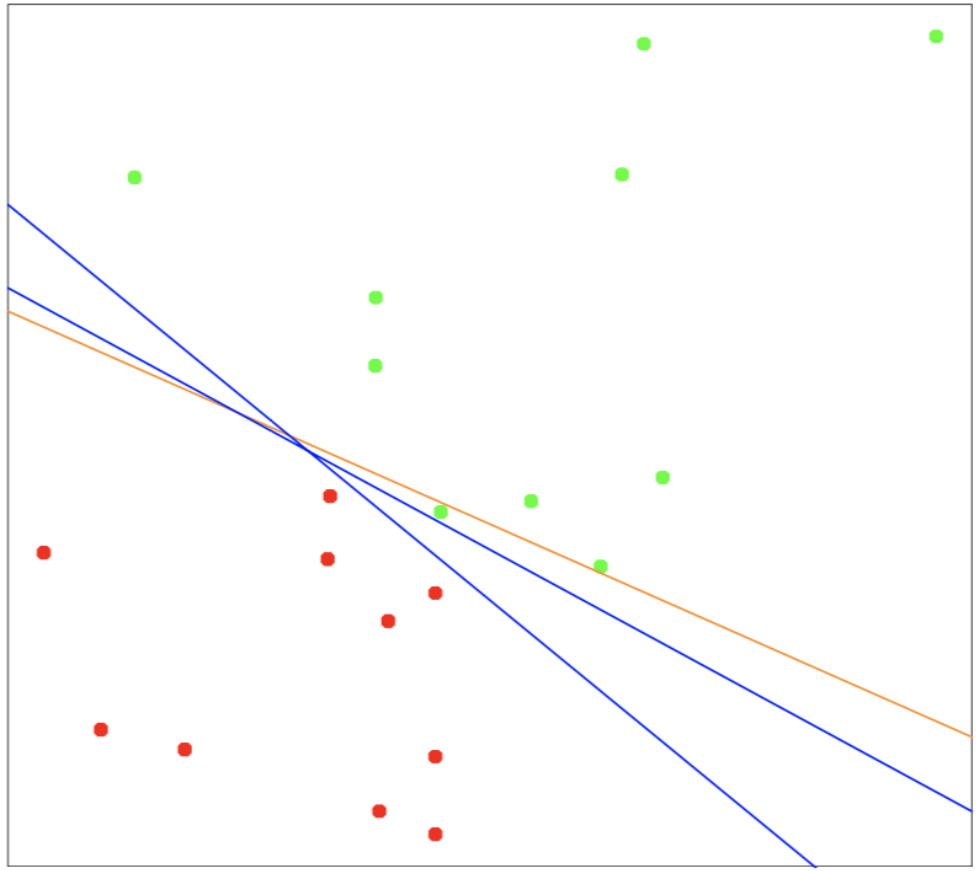


Support Vector Machines



Further reading:
Chapter 12.2, 12.3

Separating hyperplanes



Separating hyperplanes

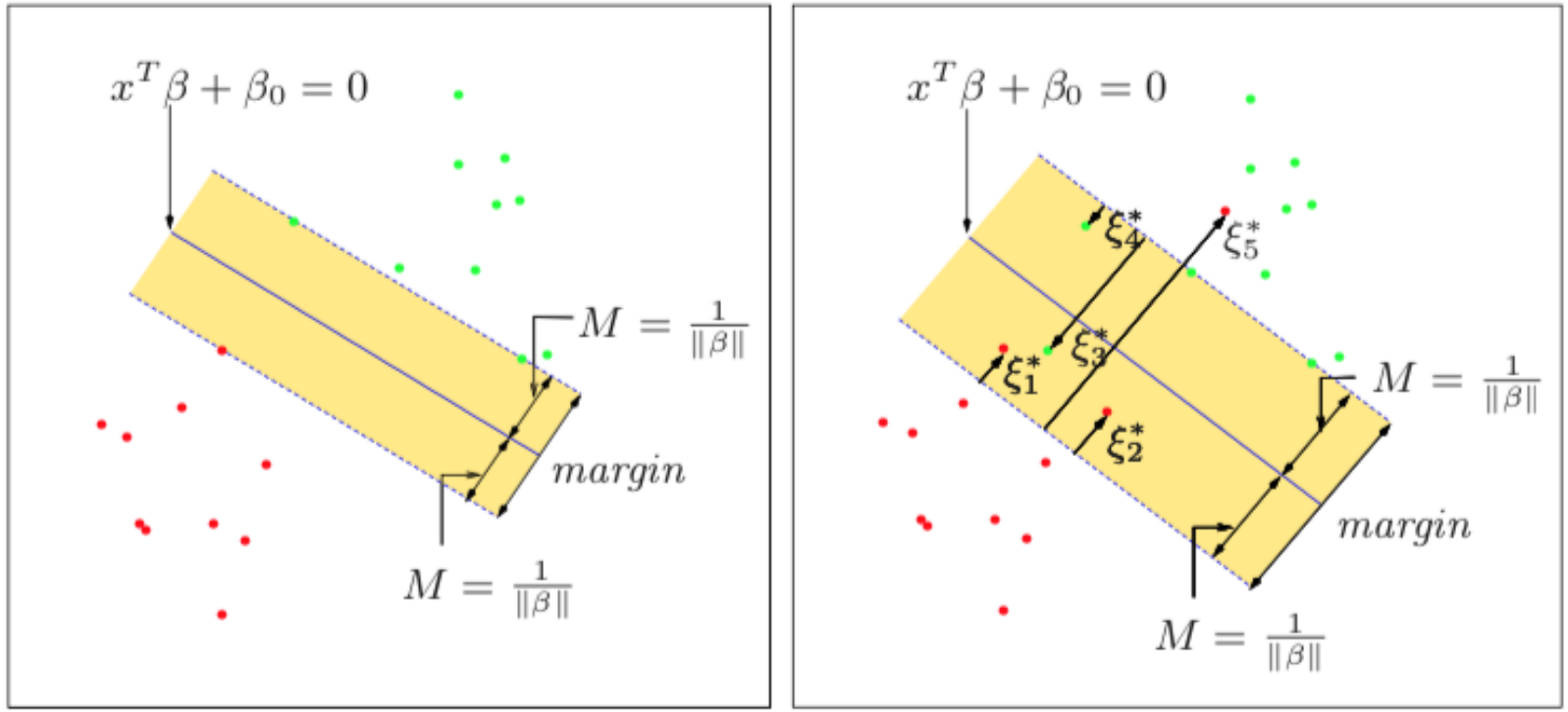


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width* $2M = 2/\|\beta\|$.

Popularity of SVMs

- $L_D(\alpha) = \sum_i^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{k=1}^N \alpha_i \alpha_k y_i y_k x_i^T x_k$
- Training and application of SVMs requires only the inner product (dot-product) of samples: $x^T x_i$ or $< x, x_i >$

- This has two advantages:
  - Cheap computation of *basis expansions*
    - $h(x) = (h_1(x), h_2(x), \dots, h_M(x))$ e.g., $h_1(x) = x_1^2, \dots$
  - We only require a similarity measure between two samples!
    - A basis expansion might not even exist!

Kernels

$$L_D(\alpha) = \sum_i^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{i'=1}^N \alpha_i \alpha_{i'} y_i y_{i'} < h(x_i), h(x_{i'}) >$$

And in prediction:

$$f(x) = h(x)^T \beta + \beta_0 = h(x)^T \sum_{i=1}^N \hat{\alpha}_i y_i h(x_i) + \beta_0$$

$$= \sum_i^N \hat{\alpha}_i y_i < h(x), h(x_i) > + \beta_0$$

In fact, we do not have to compute the dot product, we just need the Kernel function:

$$K(x, x') =< h(x), h(x') >$$

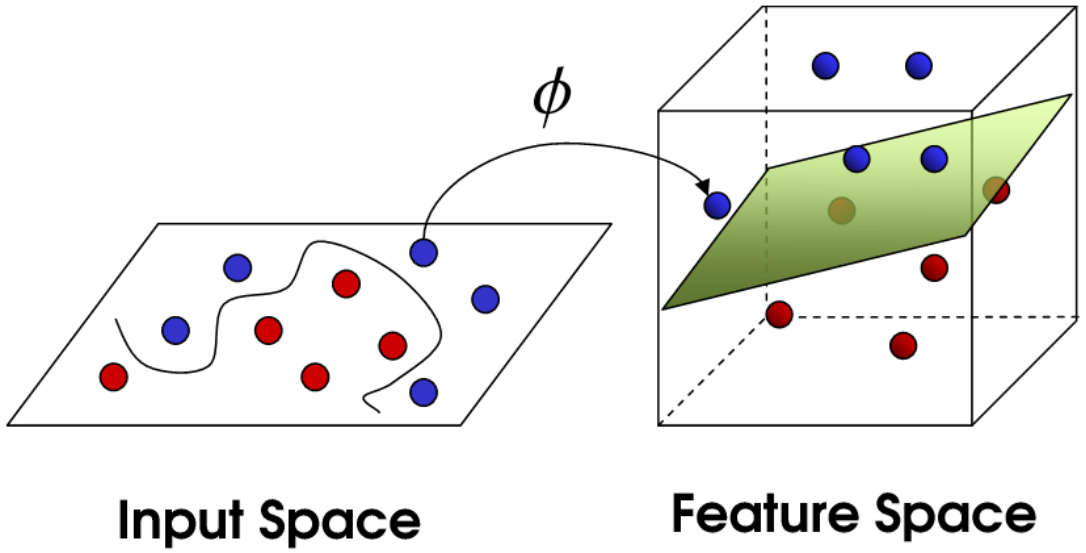
Kernel '*trick*'

x and x'

Compute dot-product

$< x, x >$

Apply $h(x)$ to each sample



Apply transformation

$h(x)$ and $h(x')$

Compute dot-product

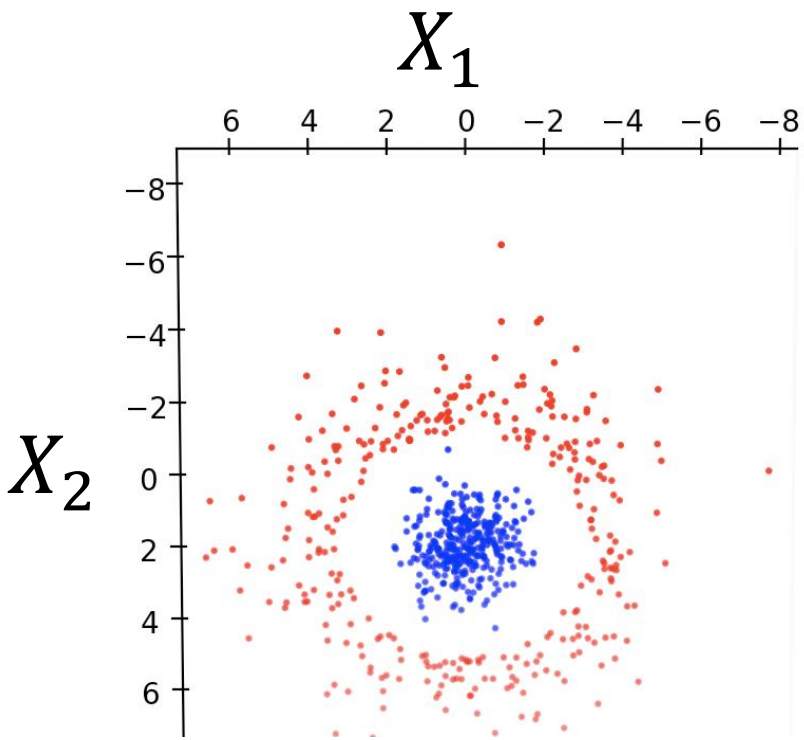
$K(h(x), h(x'))$

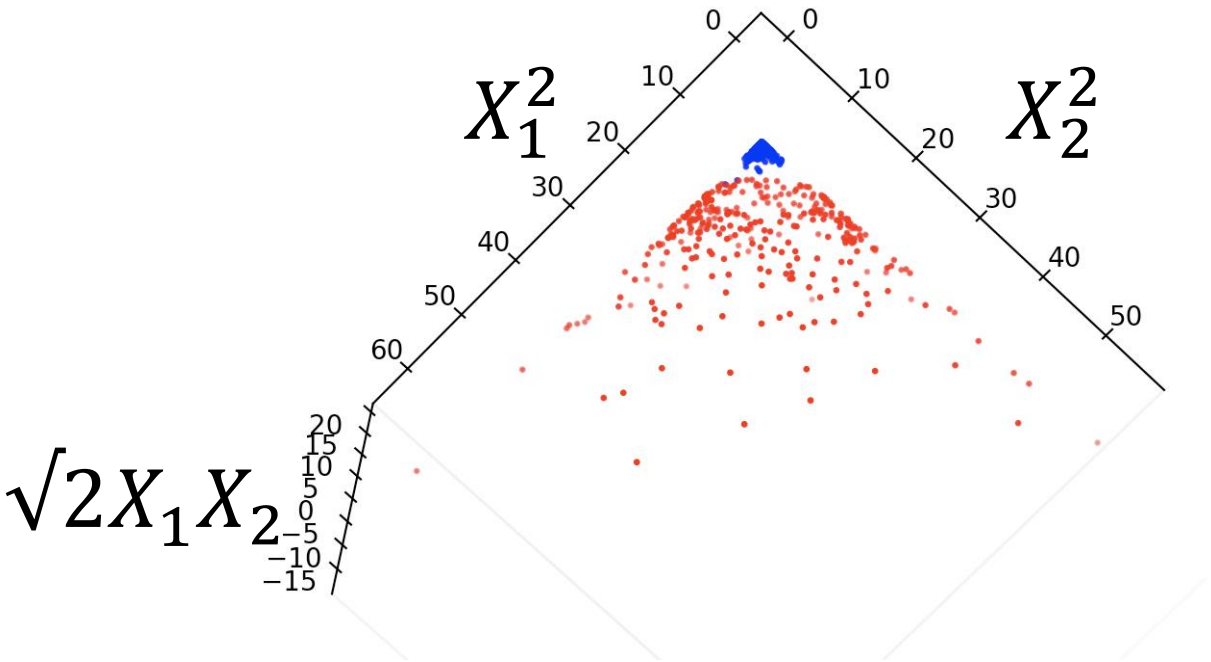
Example

- 2-dim feature space X_1 and X_2
- We want to add squares and interactions
- It is sufficient to set: $K(x, x') =< x, x' >^2$

- $$\begin{aligned} < x, y > ^2 &= (x_1 y_1 + x_2 y_2)^2 \\ &= (x_1^2 y_1^2 + 2 x_1 y_1 x_2 y_2 + x_2^2 y_2^2) \\ &= (h_1(x) h_1(y) + h_2(x) h_2(y) + h_3(x) h_3(y)) \\ &=< \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right), (y_1^2, \sqrt{2} y_1 y_2, y_2^2) > \end{aligned}$$
- $h_1(x) = x_1^2; h_2(x) = \sqrt{2} x_1 x_2; h_3(x) = x_2^2$

Example: circle data





Other kernels

- In computational biology:
  - *String kernels*

IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDY**LQE**FLGVMNTEWI

PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAER**LQE**NLQAYRTFHVLLA
RLLEDQQVHFTPTEGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK
LWGLKV**LQE**LSQWTVRSIHDLRFISSHQTGIP

- Sufficient to compute/store similarity between samples
  - E.g., from image registration

Similarity to regularized regression

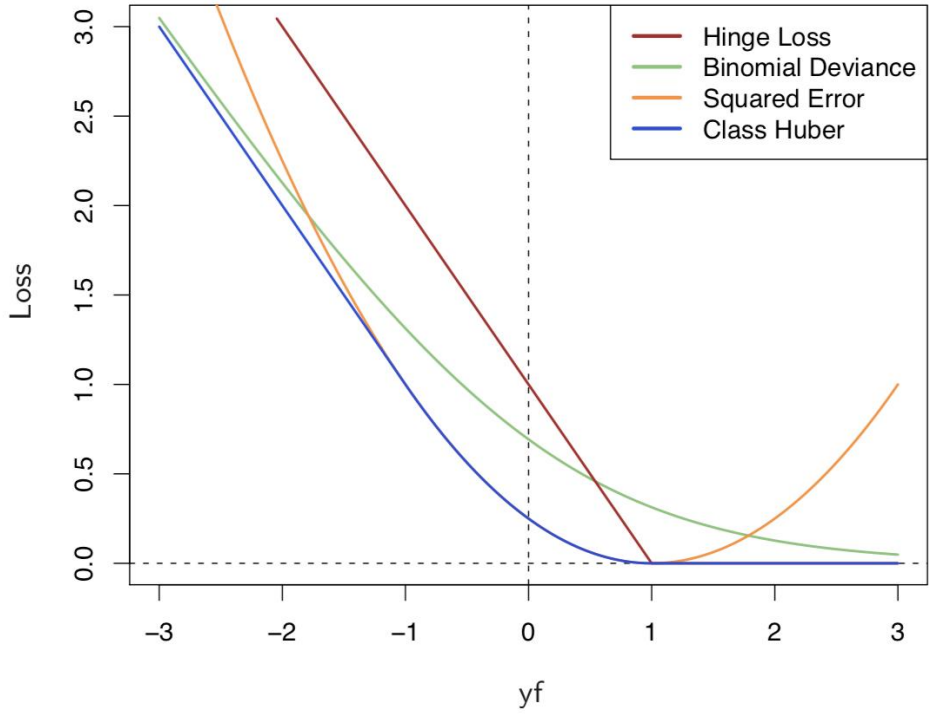
- The optimization function for SVMs can be expressed for $f(x) = h(x)^T\beta + \beta_0$ as:

$$\min_{\beta} \sum_{i=1}^{N} [1 - y_i f(x)]_+ + \frac{\lambda}{2}\|\beta\|^2$$

$$\lambda = \frac{1}{C}$$

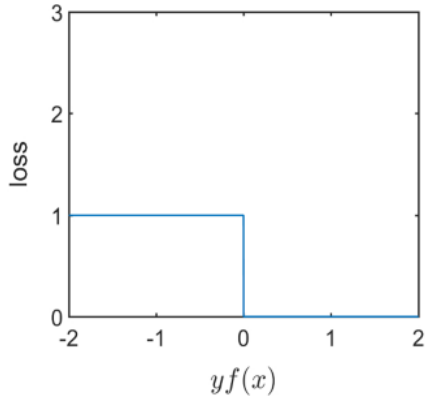
- Hinge loss: If expression in $[\quad]_+$ is negative: set to 0.

- Similar to Logistic Regression!

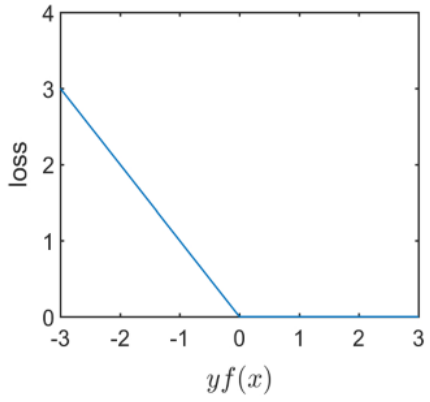


Introducing Margin

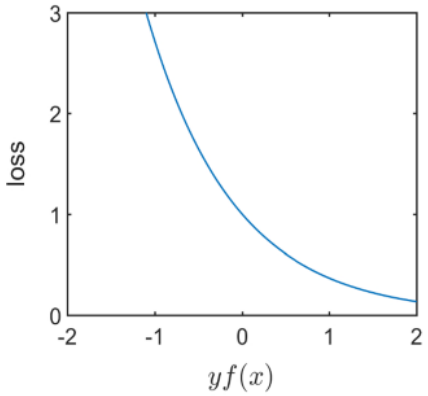
- E.g., in classification



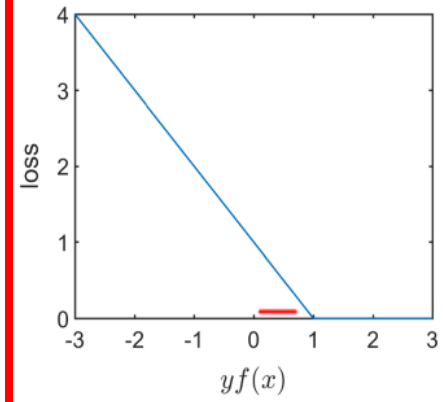
(a) 0-1 loss



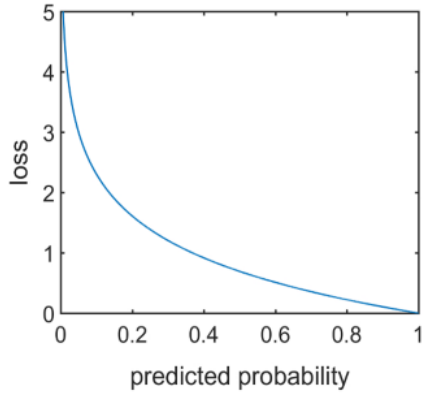
(b) perceptron loss



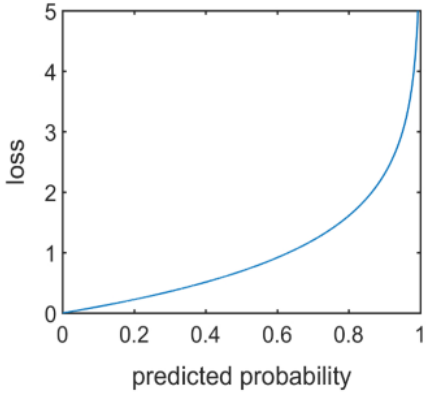
(e) exponential loss



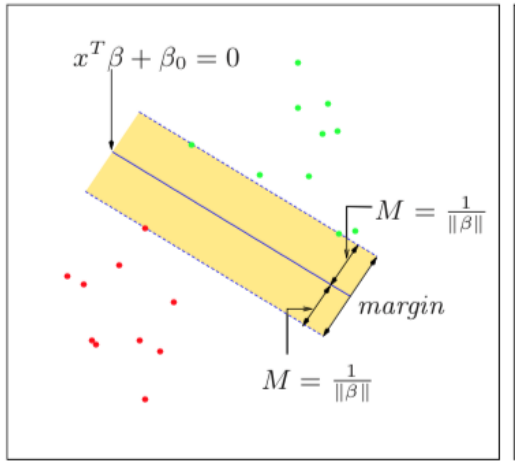
(f) hinge loss



(c) logarithmic loss(label=1)



(d) logarithmic loss(label=-1)

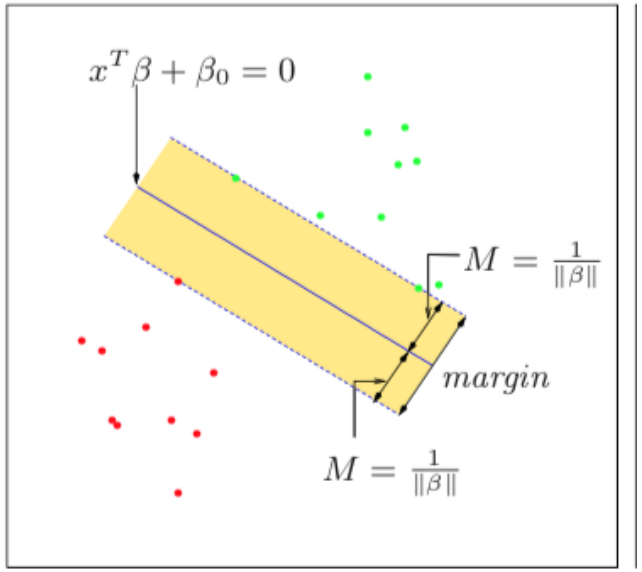


SVMs

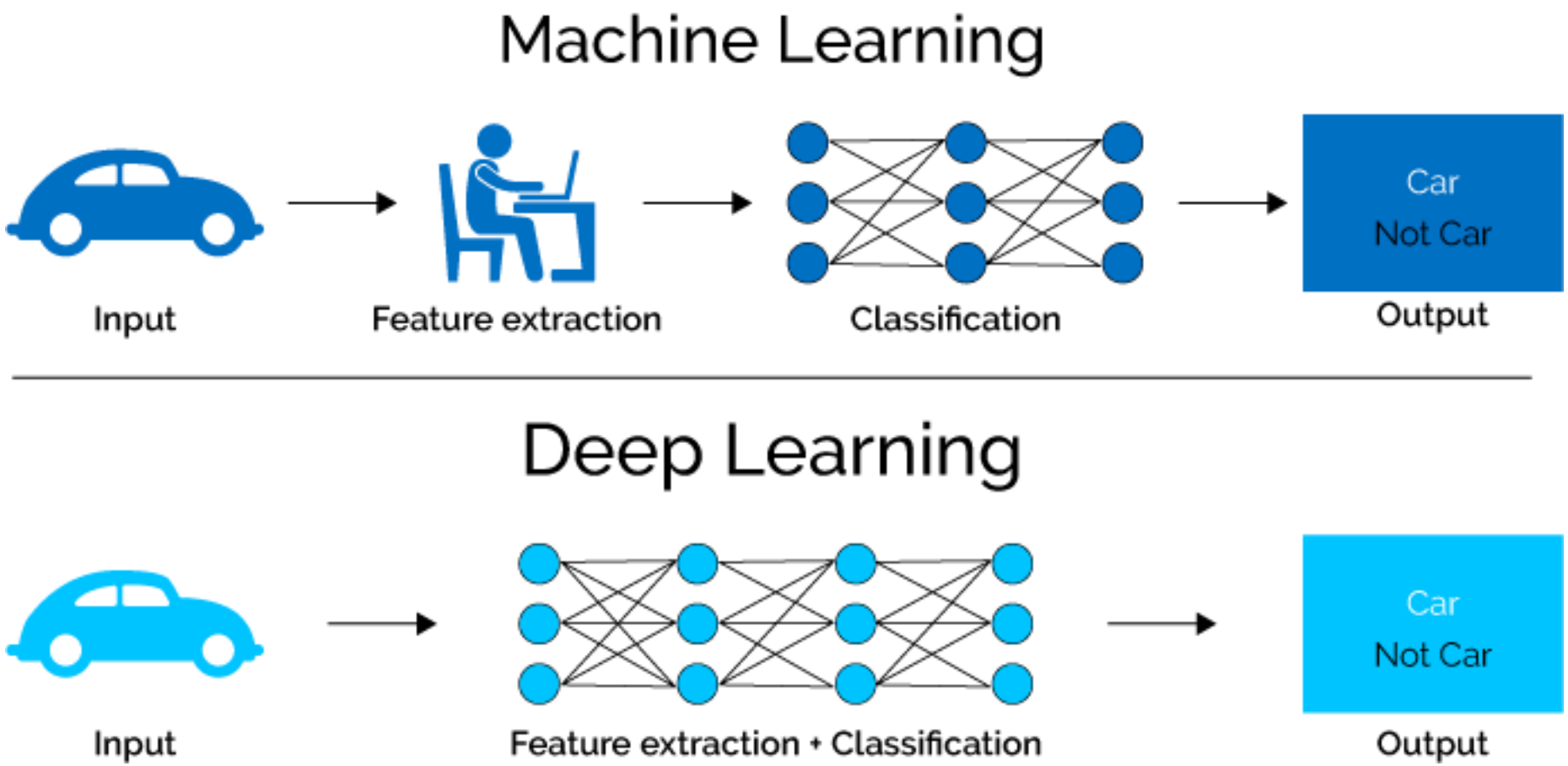
- Cope ‘well’ with high dimensional features

- Work well on small datasets

- Training scales badly with increasing N
  - $L_D(\alpha) = \sum_i^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{k=1}^N \alpha_i \alpha_k y_i y_k x_i^T x_k$

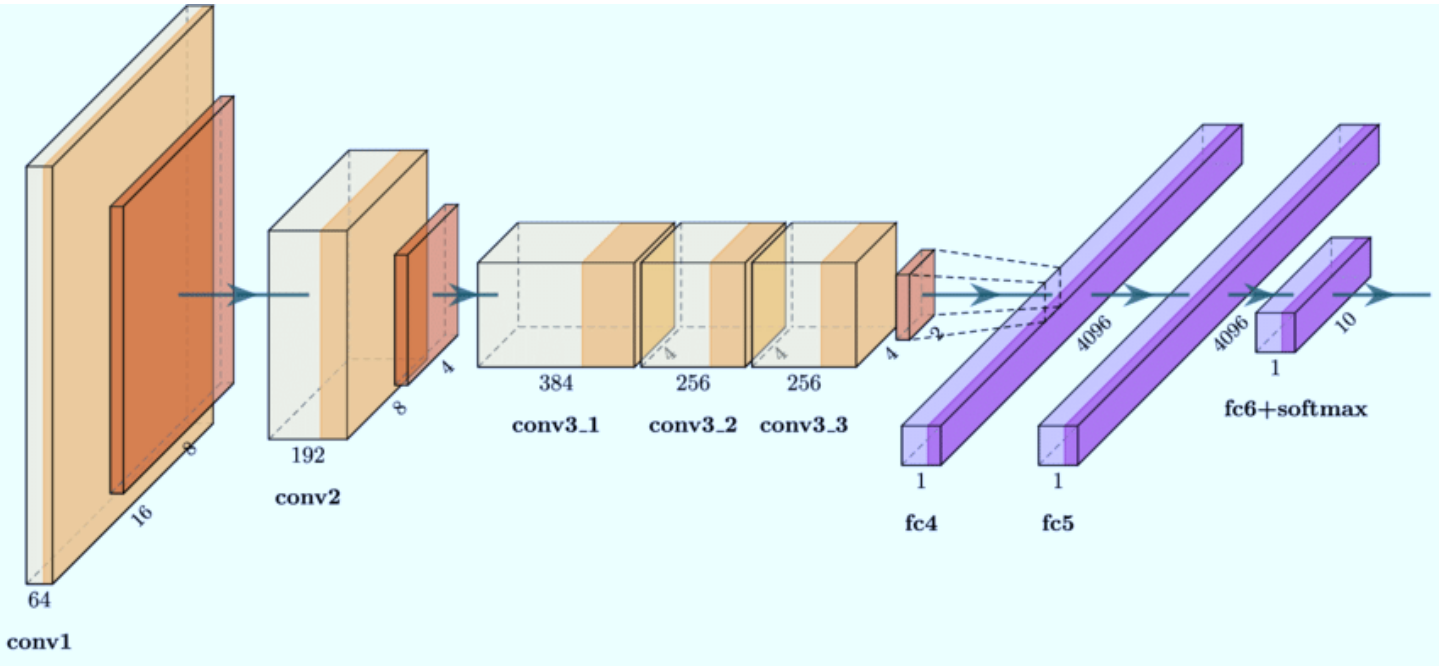


Learn features on other (unlabeled) data

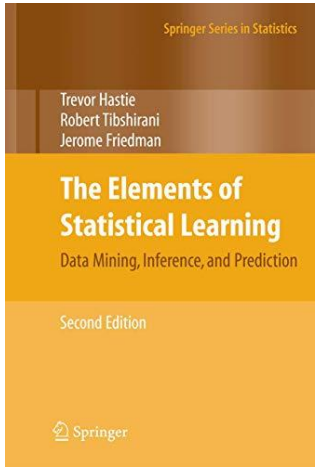

Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

Sparse Data

- Deep Learning: Use Pretrained Methods
  - Transfer Learning
  - ”Keep Feature Extraction, Learn Classification”



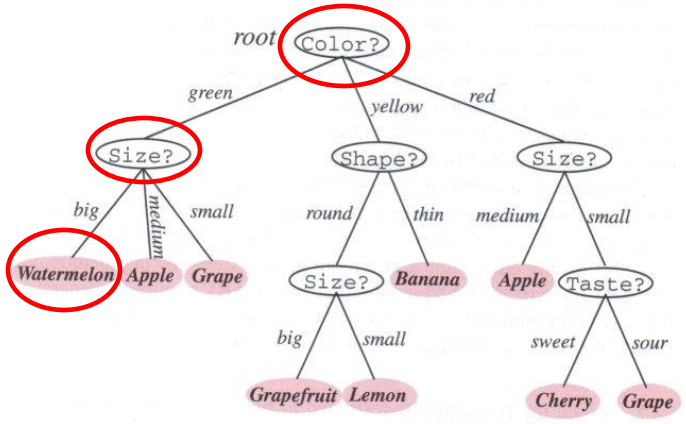
Tree-Based Methods

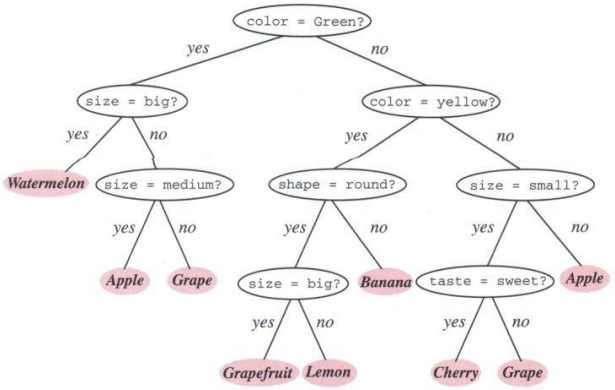


Further reading:
Chapter 9.2

Decision Trees

- Tree-based models can be used for classification and regression

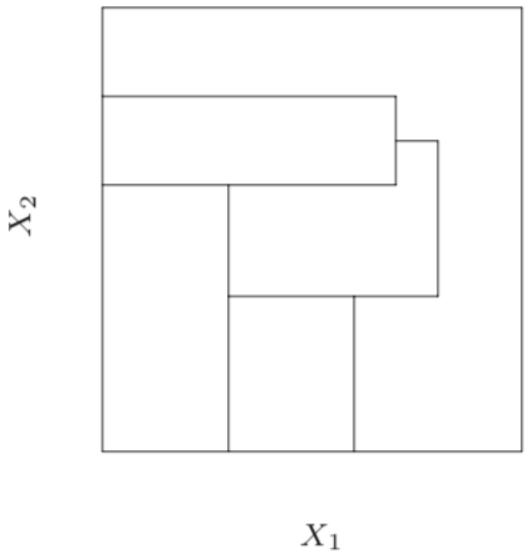




- Categorical features are supported natively
- Every tree can be displayed as a binary tree
- The main work is to decide which property test or *query* should be performed at each node

Tree-based models

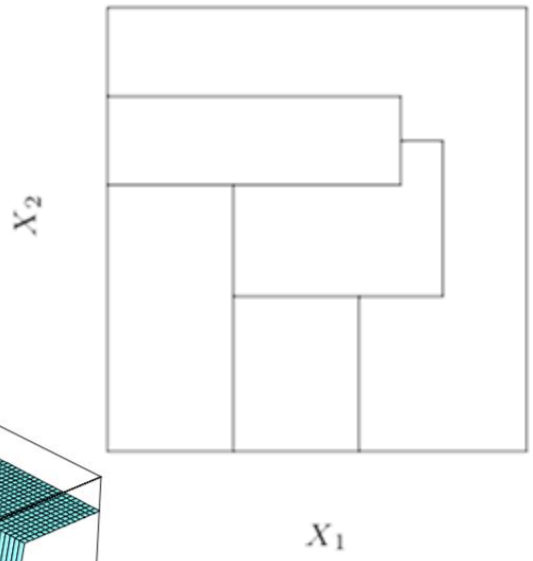
- Partition the features space
  - Example: X_1 and X_2

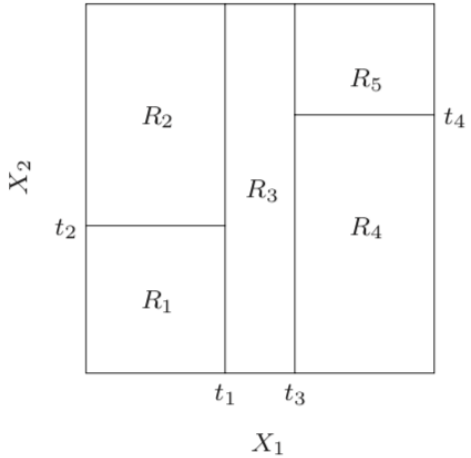


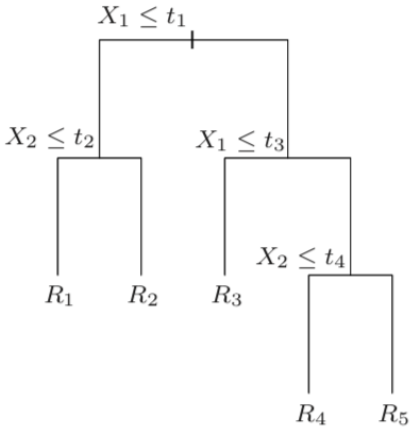
- <u>Classification:</u> each region R_i receives a class label
- <u>Regression:</u> each region a R_i receives a continuous value
- Regions defined by *cuts* along the feature axes e.g., $X_1 > t$

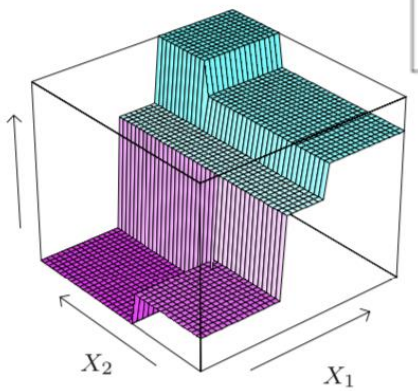
Tree-based models

- Decision boundaries are complex to describe
- Simplification: recursive binary splits









- Formally: $\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}$

Tree-based models

- Popular:
  - Interpretable in higher dimensions

difficult to draw, but the binary tree representation works in the same way. This representation is also popular among medical scientists, perhaps because it mimics the way that a doctor thinks. The tree stratifies the

- How to grow (i.e., train) a tree?

Ensembles

Ensembles

- Definition of Ensembles
  - Combination of Different Models
    - Variation of the method (*f*)
    - Variation of features (*views*)
    - Variation of training data
  - Methods vary in forming the consensus
    - Majority Vote
    - Averaging
    - Weighted Averaging
    - …
- Why? Different models have different strengths

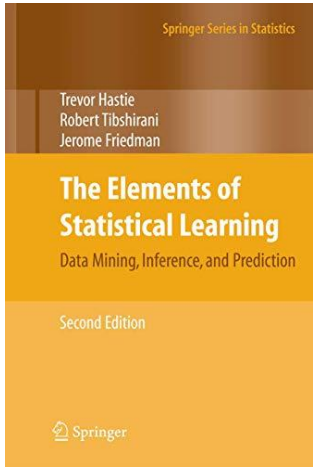
Bagging

- Bootstrap aggregation = *bagging*
- Improve parameter estimate or prediction

- Training data $\mathbf{Z} = \{(x_1, y_1), \dots, (x_N, y_N)\}$
- Regression: $\hat{f}(x)$ at input x

- Bootstrap samples $\mathbf{Z}^{*b}, b = 1, \dots, B \rightarrow \hat{f}^{*b}(x)$

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$



Further reading: Chapter 8.7

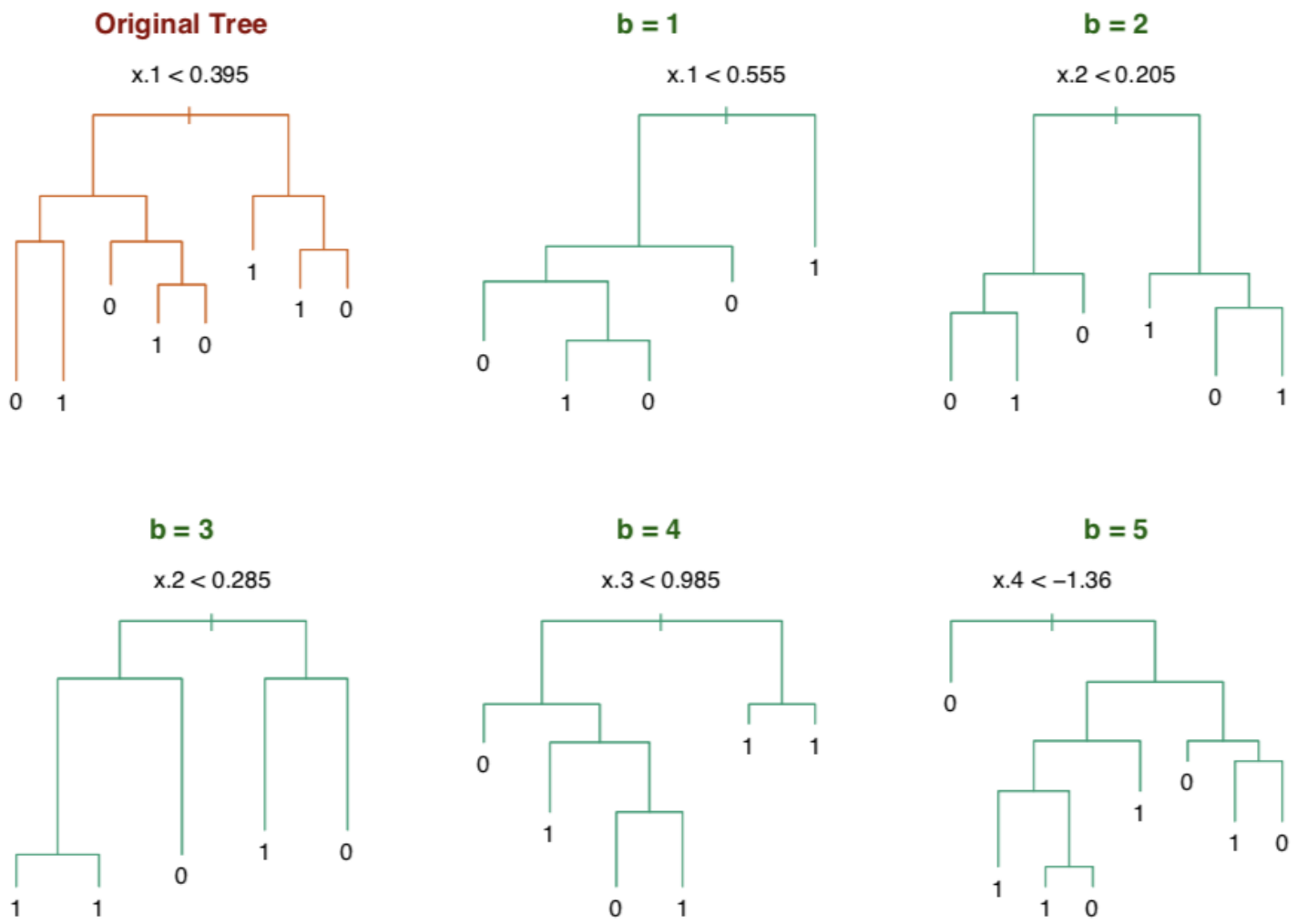
Bagging – II

- If $f(x)$ is linear, then $\hat{f}(x) = \hat{f}_{\text{bag}}(x)$
- Requires a non-linear method (e.g., trees)

- Classification trees for K class response: $\hat{G}(x)$ (output is 1-hot encoding e.g.: $(0\ 0\ 1\ 0)$
- Output of $\hat{f}_{\text{bag}}(x)$ is $(p_1(x)\ \ p_2(x)\ \ p_3(x)\ \ p_4(x))$

- $\hat{G}_{bag}(x) = \text{argmax}_k \hat{f}_{\text{bag}}(x)$ "majority vote"
- *Wisdom of crowds*

Tree models


Original Tree
x.1 < 0.395
b = 1
x.1 < 0.555
b = 2
x.2 < 0.205
b = 3
x.2 < 0.285
b = 4
x.3 < 0.985
b = 5
x.4 < −1.36

Bagging – III

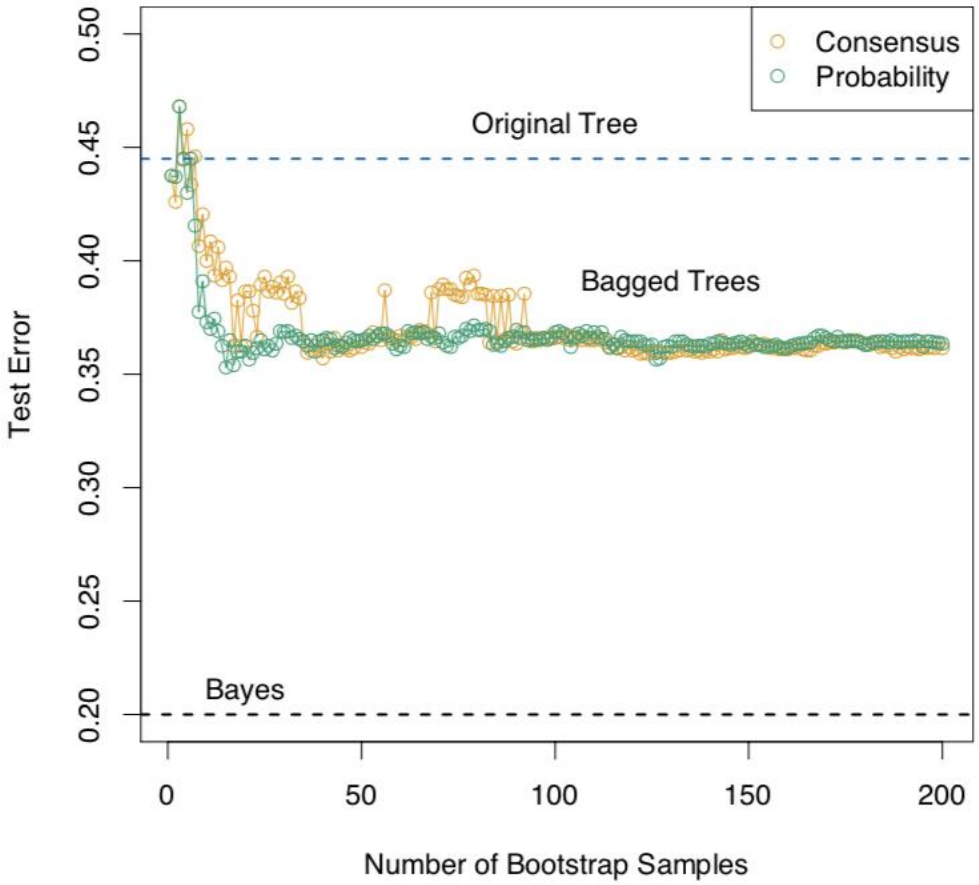
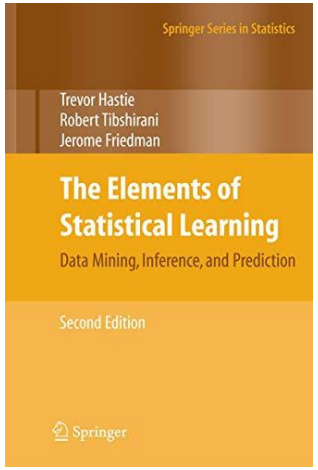


FIGURE 8.10. *Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.*

Random Forest



Further reading:
Chapter 15

Random Forest

- Forest = many trees
- Based on ‘bagging’ rather than boosting
- Simpler to train and tune than boosting

Next Part

- Idea:
  - Grow trees on bootstraps of the data $\{\mathbf{Z}^{*1}, \mathbf{Z}^{*2}, \ldots, \mathbf{Z}^{*B}\}$
  - Independently and de-correlated
  - Performance increases due to reduction in variance (bias remains)
  - In boosting: performance gain through reduction of bias

Bias-variance-tradeoff

- One can decompose MSE into Bias and variance

$$\mathrm{MSE} = \mathbb{E}\left[\left(\hat{\theta}_m - \theta\right)^2\right]$$
$$= \mathrm{Bias}\left(\hat{\theta}_m\right)^2 + \mathrm{Var}(\hat{\theta}_m)$$

- Bias: "Error made by assumption."
- Variance: "Error made b/c adopting to data."

Random Forest – II

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

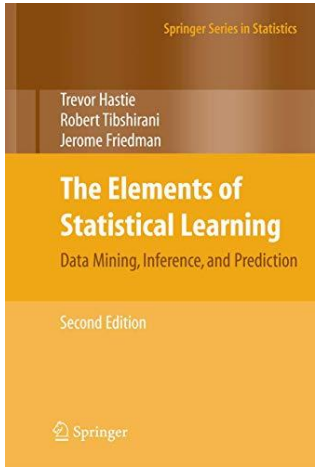
Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.

Random Forest – III

- Usually don't overfit, i.e., more trees don't hurt
- Not a 'Black Box'
  - Can derive variable importance (like the βs)
  - E.g., Mean decrease in Gini Index, mean increase in accuracy
- Get an error estimate 'for free' (out of bag error)
- Typically performs quite well

Boosting



Further reading:
Chapter 10.1

Boosting

- Very powerful learning idea “in the last 30 years” (well,… before deep learning…)
- Originally designed for classification (works for regression as well)
- Motivation: combined “weak” classifiers to produce a powerful “committee”

AdaBoost.M1

- Two class problem $Y \in \{-1, 1\}$
- Predictors X
- Classifier $G(X)$
- Training error: $\overline{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i))$

- A "*weak*" classifier performs slightly better than random guessing
- In boosting, a weak classification algorithm is applied to repeatedly modified version of the data

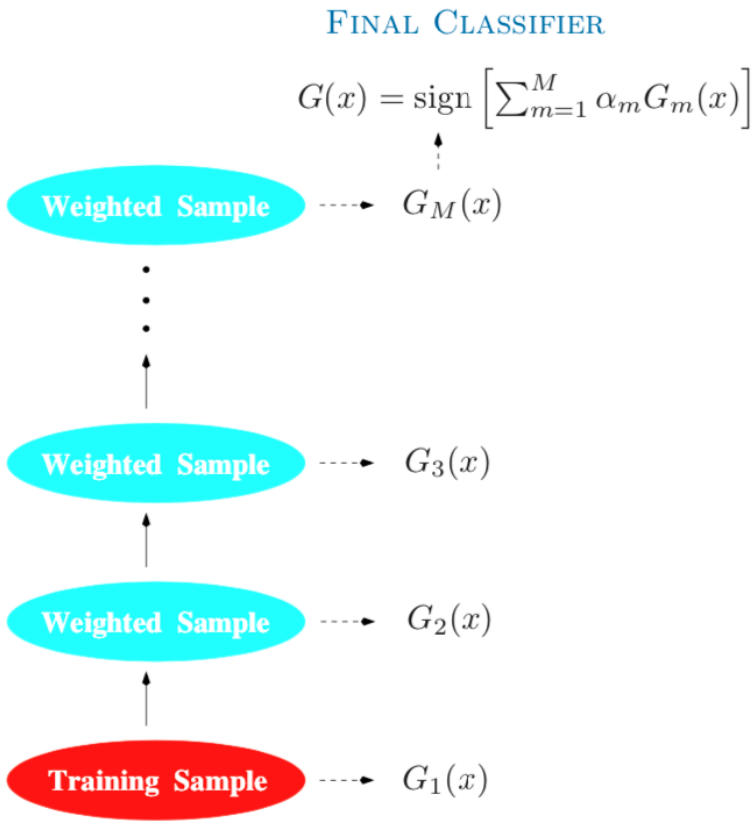
$$G_m(x), m = 1, 2, \ldots, M$$

AdaBoost.M1

- Predictions are formed through a weighted majority vote

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

- The weights α_m are derived by the *boosting* algorithm



GURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted ver-*
ns of the dataset, and then combined to produce a final prediction.

AdaBoost.M1

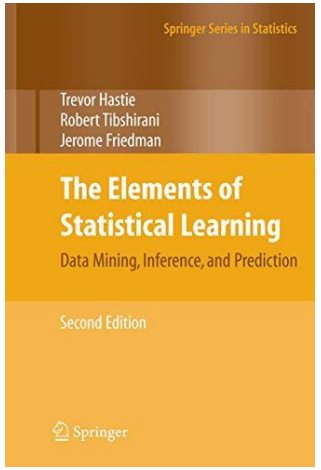
- AdaBoost *modifies* the data at each iteration
- It uses weights $w_1, w_2, \dots, w_N$ for each sample
  - Initialized with $w_i = \frac{1}{N}$
- $G_1(x)$ is trained on original dataset
- For the following iterations $m = 2, \dots, M$ each sample is re-weighted
- At each step m the samples that were misclassified by $G_{m-1}(x)$ receive a higher weight
  - Thus: we try harder to classify them correctly
  - Weights for correctly classified samples will be decreased

AdaBoost.M1

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   Weighted error
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$. Classifier weight
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$. Sample weight
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Gradient Boosted Trees

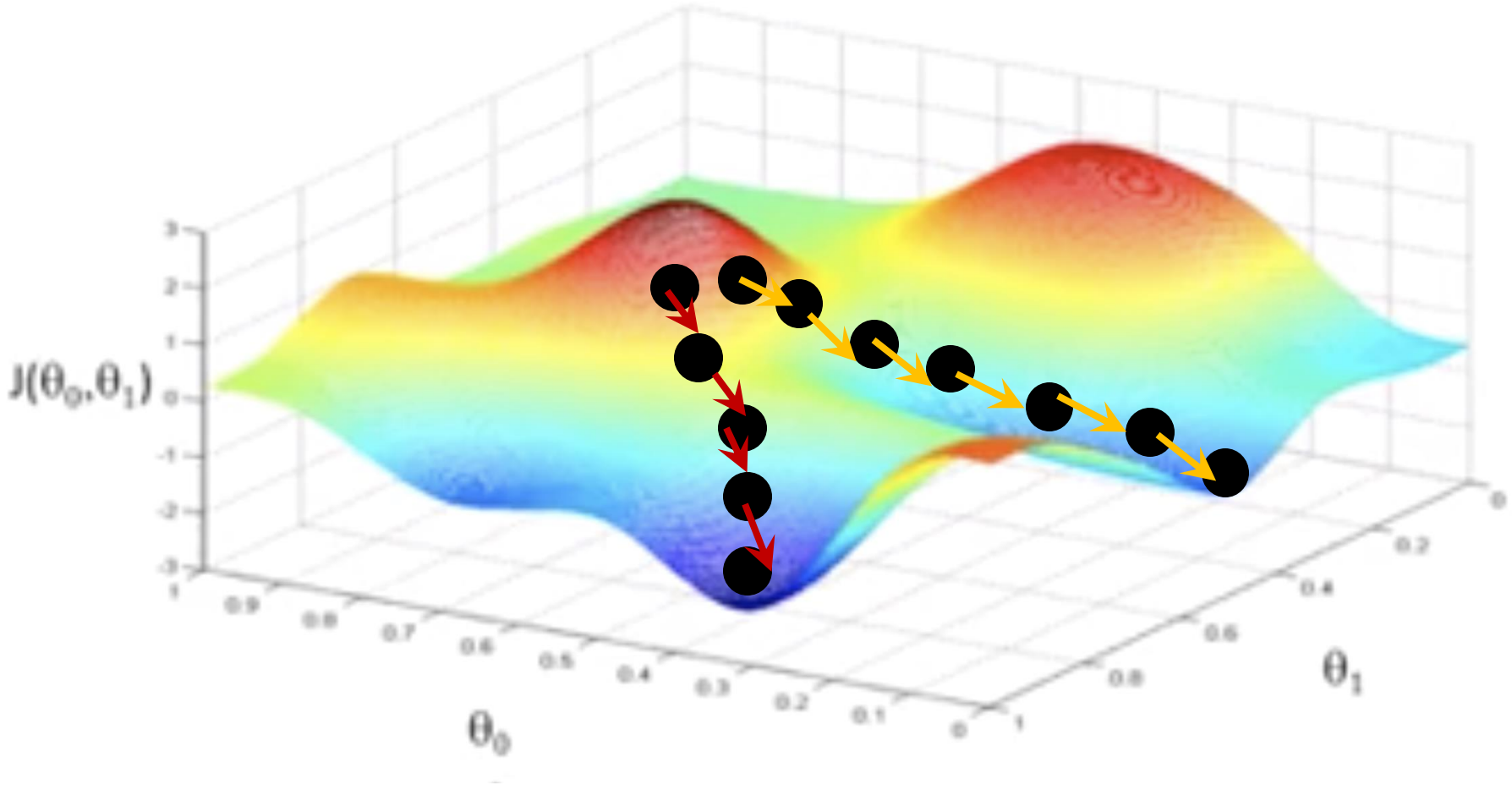


Further reading:
Chapter 10.10-10.12

Gradient Boosting

- First: Gradient decent: $J(\theta_0, \theta_1)$

$$\theta_i = \theta_i - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_i}$$



Gradient Boosting – II

- We can use the same principle to find the function $f(x)$
- Loss on our dataset: $L(f) = \sum_i L\big(y_i, f(x_i)\big)$
- Same as finding a vector $\hat{\mathbf{f}} = \operatorname{argmin}_{\mathbf{f}} L(\mathbf{f})$, where
$$\mathbf{f} = \{f(x_1), f(x_2), \dots, f(x_N)\}^T$$
- Using numerical optimization methods one can find:
$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m$$
- Where $\mathbf{h}_0$ is an initial guess and $\mathbf{h}_m$ are steps

Gradient Boosting – III

- One option: steepest descent (like gradient descent):

$$\mathbf{h}_m = -\rho_m \mathbf{g}_m$$

- The g_m are gradients for sample x_i:

$$g_{im} = \left[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}$$

- And ρ_m is the step length
- Simplifies to: $\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$

From earlier:

$$\theta_i = \theta_i - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_i}$$

Gradient Boosting – IV

- The gradient $\mathbf{g}_m$ is <u>only defined for training data</u>, but we want a model that makes predictions for new data.
- Approximate $\mathbf{g}_m$ with a tree!
  - Add the tree that reduces the loss the most (given the current model)
  - Like gradient descent (step along the gradient)
- Conceptually: $-\rho_m \mathbf{g}_m \approx t_m = \{T(x_1, \Theta_m), \dots, T(x_N, \Theta_m)\}^T$

Θ_m: cut points and regional assignments for tree *m*

$$\widehat{\Theta}_m = \operatorname{argmin} \sum_i^N \left(-g_{im} - T(x_i; \Theta)\right)^2$$

Gradient Boosting – V

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\boxed{\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]}_{f=f_{m-1}}$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

For squared error loss it is just the residual error!

GBT

- Gradient Boosted Trees are the go-to for tabular data

TABULAR DATA: DEEP LEARNING IS NOT ALL YOU NEED

Ravid Shwartz-Ziv
ravid.ziv@intel.com
IT AI Group, Intel

Amitai Armon
amitai.armon@intel.com
IT AI Group, Intel

November 24, 2021

Dataset	Features	Classes	Samples	Source	Paper
Gesture Phase	32	5	9.8k	OpenML	DNF-Net
Gas Concentrations	129	6	13.9k	OpenML	DNF-Net
Eye Movements	26	3	10.9k	OpenML	DNF-Net
Epsilon	2000	2	500k	PASCAL Challenge 2008	NODE
YearPrediction	90	1	515k	Million Song Dataset	NODE
Microsoft (MSLR)	136	5	964k	MSLR-WEB10K	NODE
Rossmann Store Sales	10	1	1018K	Kaggle	TabNet
Forest Cover Type	54	7	580k	Kaggle	TabNet
Higgs Boson	30	2	800k	Kaggle	TabNet
Shrutime	11	2	10k	Kaggle	New dataset
Blastchar	20	2	7k	Kaggle	New dataset

Table 1: Description of the tabular datasets

GBT

Model Name	Rossman	CoverType	Higgs	Gas	Eye	Gesture
XGBoost	490.18 ± 1.19	3.13 ± 0.09	21.62 ± 0.33	2.18 ± 0.20	**56.07**±0.65	80.64 ± 0.80
NODE	488.59 ± 1.24	4.15 ± 0.13	21.19 ± 0.69	2.17 ± 0.18	68.35 ± 0.66	92.12 ± 0.82
DNF-Net	503.83 ± 1.41	3.96 ± 0.11	23.68 ± 0.83	**1.44** ±0.09	68.38 ± 0.65	86.98 ± 0.74
TabNet	**485.12**±1.93	3.01 ± 0.08	**21.14**±0.20	1.92 ± 0.14	67.13 ± 0.69	96.42 ± 0.87
1D-CNN	493.81 ± 2.23	3.51 ± 0.13	22.33 ± 0.73	1.79 ± 0.19	67.9 ± 0.64	97.89 ± 0.82
Simple Ensemble	488.57 ± 2.14	3.19 ± 0.18	22.46 ± 0.38	2.36 ± 0.13	58.72 ± 0.67	89.45 ± 0.89
Deep Ensemble w/o XGBoost	489.94 ± 2.09	3.52 ± 0.10	22.41 ± 0.54	1.98 ± 0.13	69.28 ± 0.62	93.50 ± 0.75
Deep Ensemble w XGBoost	485.33 ± 1.29	**2.99** ±0.08	22.34 ± 0.81	1.69 ± 0.10	59.43 ± 0.60	**78.93** ±0.73

TabNet DNF-Net

Model Name	YearPrediction	MSLR	Epsilon	Shrutime	Blastchar
XGBoost	77.98 ± 0.11	55.43±2e-2	11.12±3e-2	13.82 ± 0.19	20.39 ± 0.21
NODE	76.39 ± 0.13	55.72±3e-2	**10.39**±1e-2	14.61 ± 0.10	21.40 ± 0.25
DNF-Net	81.21 ± 0.18	56.83±3e-2	12.23±4e-2	16.8 ± 0.09	27.91 ± 0.17
TabNet	83.19 ± 0.19	56.04±1e-2	11.92±3e-2	14.94±, 0.13	23.72 ± 0.19
1D-CNN	78.94 ± 0.14	55.97±4e-2	11.08±6e-2	15.31 ± 0.16	24.68 ± 0.22
Simple Ensemble	78.01 ± 0.17	55.46±4e-2	11.07±4e-2	13.61±, 0.14	21.18 ± 0.17
Deep Ensemble w/o XGBoost	78.99 ± 0.11	55.59±3e-2	10.95±1e-2	14.69 ± 0.11	24.25 ± 0.22
Deep Ensemble w XGBoost	**76.19** ±0.21	**55.38**±1e-2	11.18±1e-2	**13.10**±0.15	**20.18**±0.16

NODE New datasets

Deep And Classic

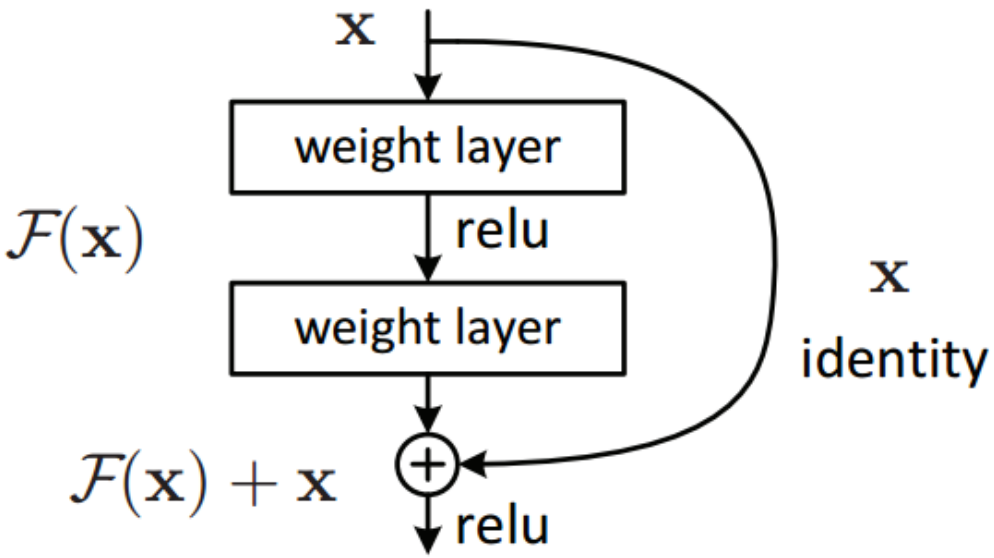
1. Need Large Data
   - Pre-trained models
   - SVM, linear models, tree ensembles
2. Data Type
   - Tree ensembles
3. Training Requires Substantial Computational Resources
   - Linear models, Tree ensembles
4. Mostly a ‘Black Box’ model
   - Interpretable models, e.g., linear models, tree-based models

Ensembles and Deep Learning

ResNet and Ensembles?

Residual Networks Behave Like Ensembles of Relatively Shallow Networks

https://arxiv.org/abs/1605.06431



$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

Skin Lesion Detection

Contents lists available at ScienceDirect

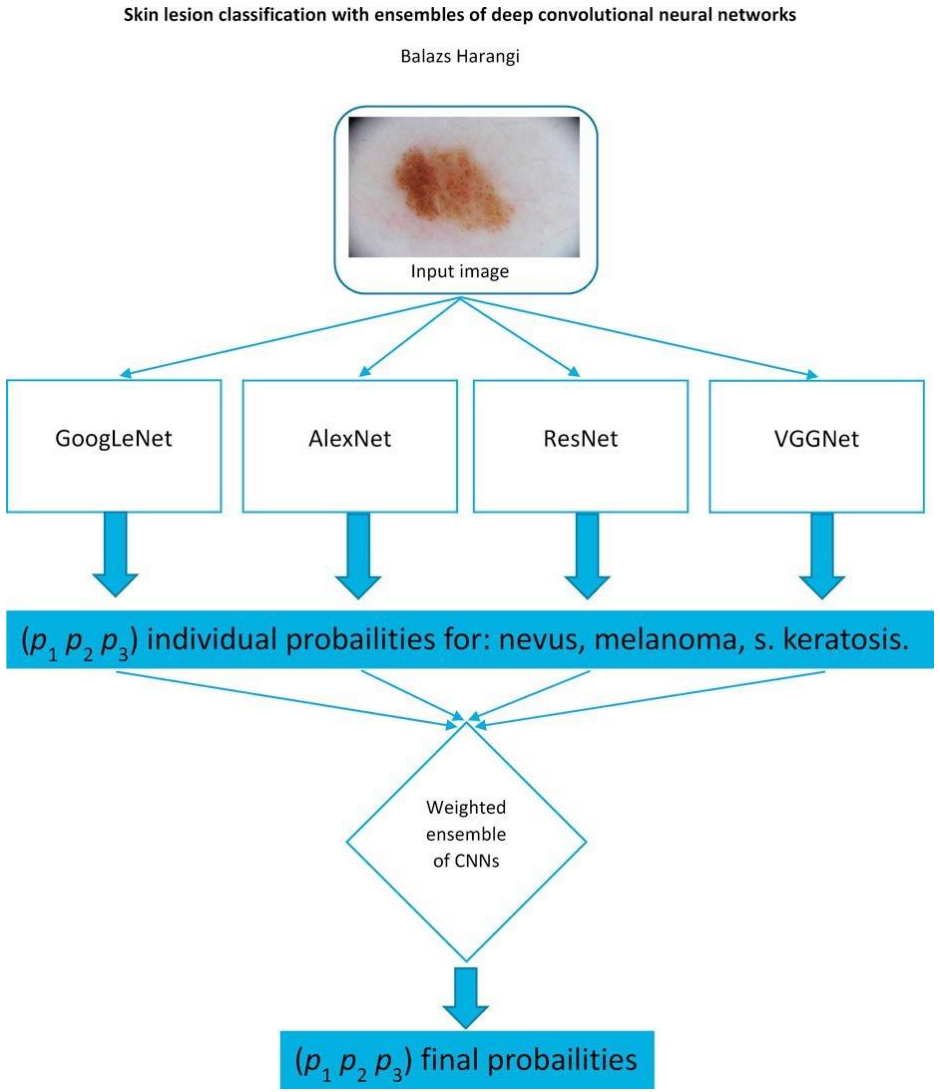
Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin

Skin lesion classification with ensembles of deep convolutional neural networks

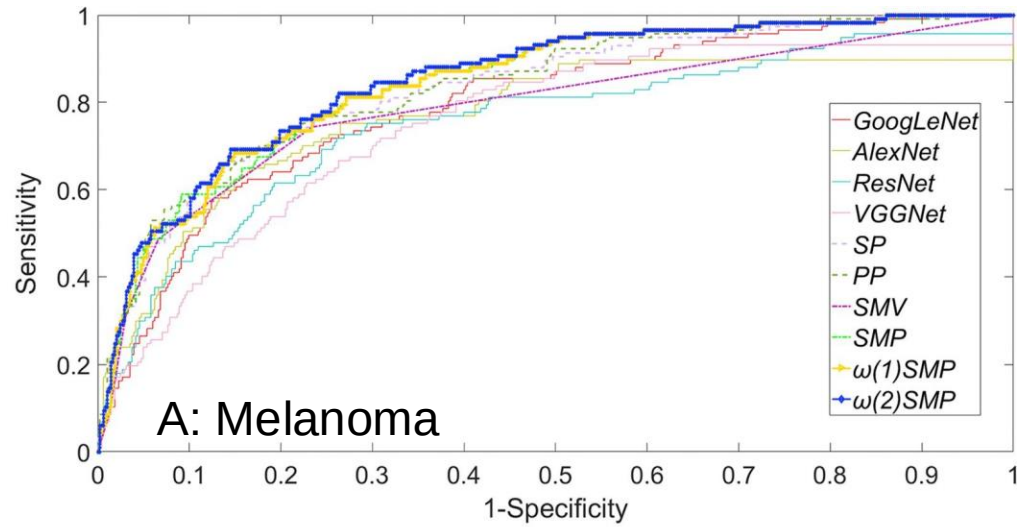
Balazs Harangi

- 3 classes
- 2000 training / 600 test images
- Standard augmentation techniques
- Transfer Learning/Fine Tuning of large CNNs
- Explore different voting schemes

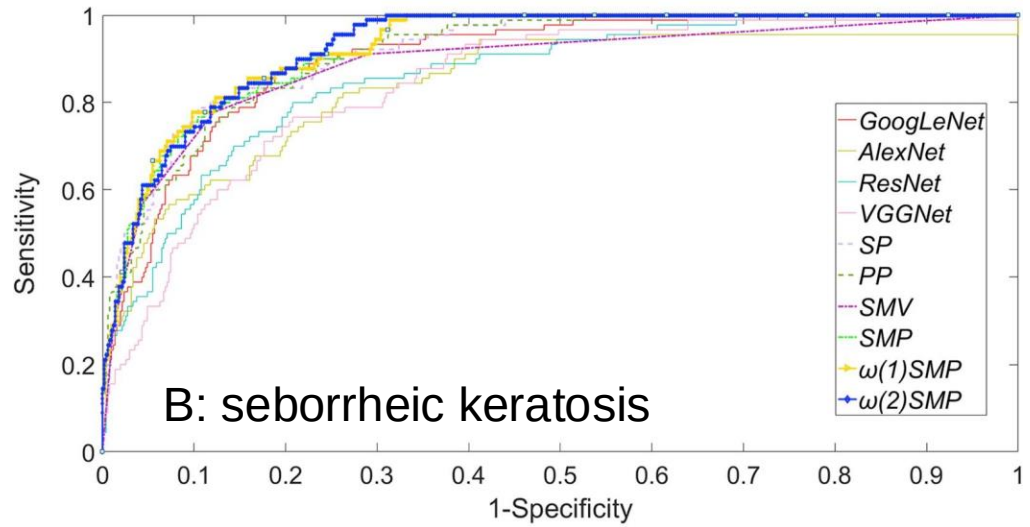


Skin Lesion Detection

- Different Fusion Approaches
  - Sum of Probabilities
  - Product of Probabilities
  - Simple Majority Voting
  - Sum of Max Probability
  - Weighted Average
- Fusion approaches outperform individual Networks
- Weighted Fusion works best (how to get the best weights?)

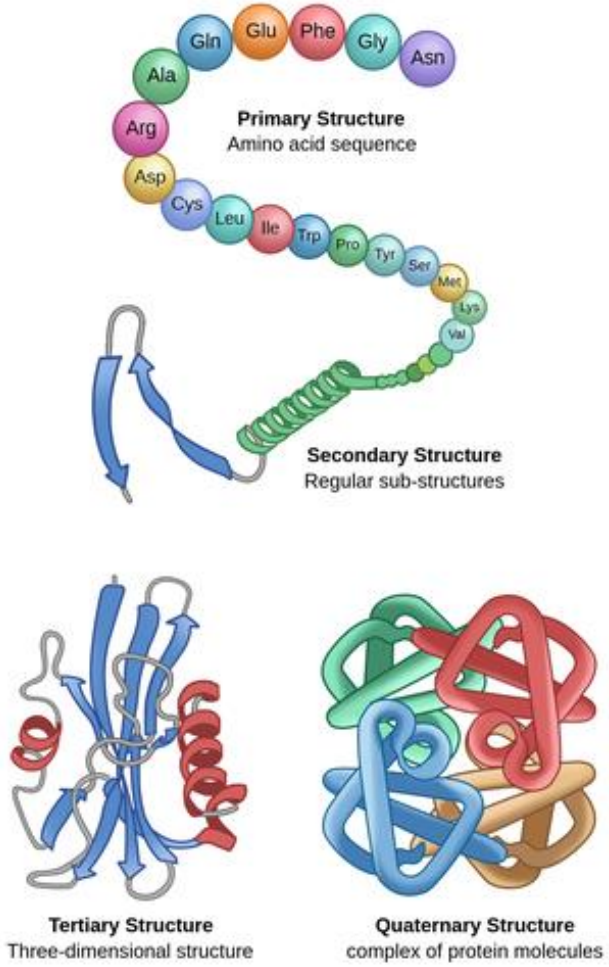


(a)



(b)

Protein-Protein Interaction Prediction



ELSEVIER

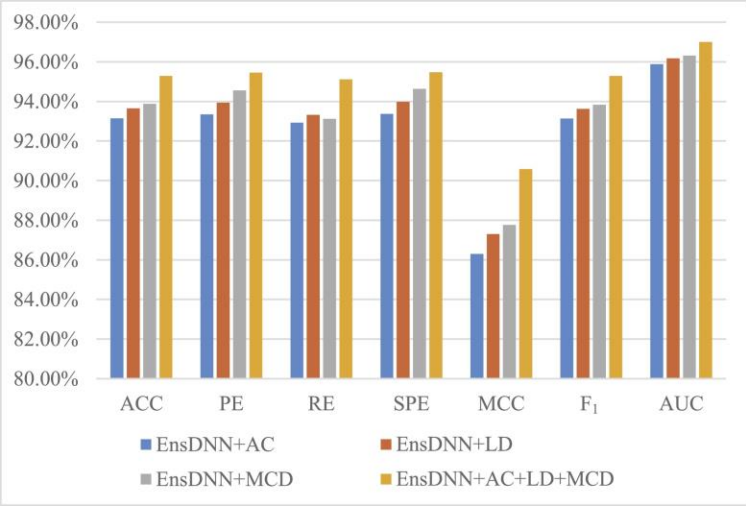
Neurocomputing
Volume 324, 9 January 2019, Pages 10-19

Protein–protein interactions prediction based on ensemble deep neural networks

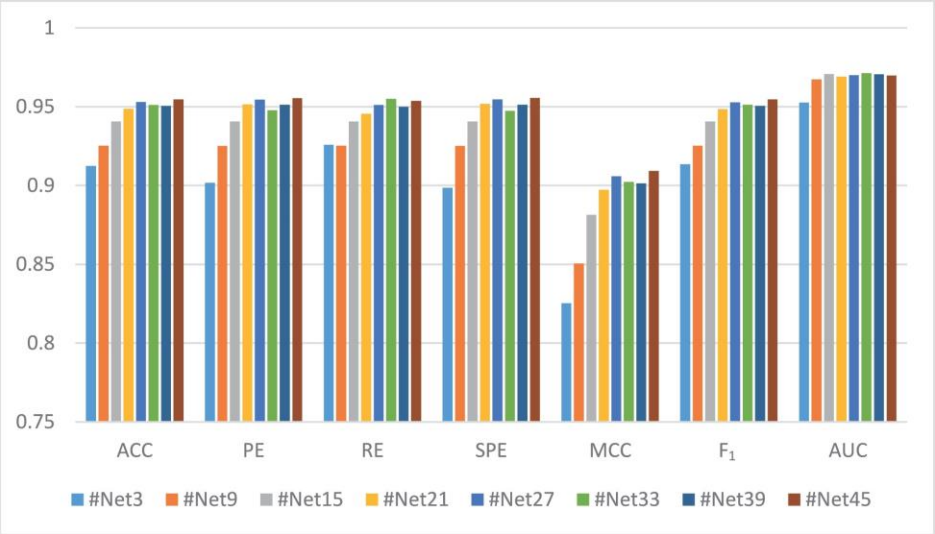
Long Zhang [a], Guoxian Yu [a], Dawen Xia [b c], Jun Wang [a]

- Different representations of amino acid sequences
- 3 different feature sets (manual)
- 9 different architectures (depth/width)
- 27 outputs are input to a 2-hidden layer MLP

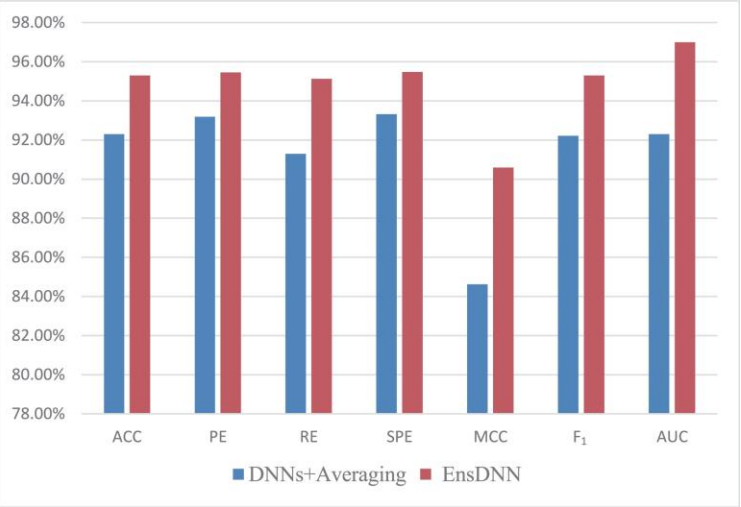
Protein-Protein Interaction Prediction



Ensembles with all features performs best



Ensembles of ~27 nets are sufficient



Weighting > Averaging

Deep Ensembles

- Randomly initialized neural networks explore different modes in function space
- Deep ensembles trained with just random initializations work well in practice
- Current variational Bayesian methods lack diversity

Deep Ensembles: A Loss Landscape Perspective

Stanislav Fort*
Google Research
sfort1@stanford.edu

Huiyi Hu*
DeepMind
clarahu@google.com

Balaji Lakshminarayanan†
DeepMind
balajiln@google.com

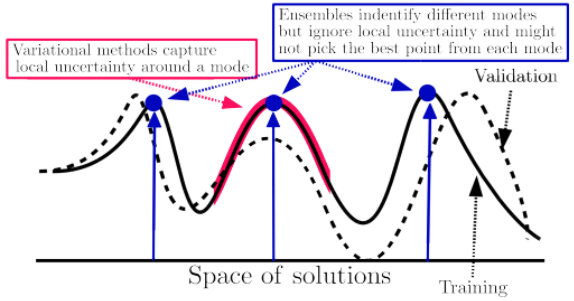
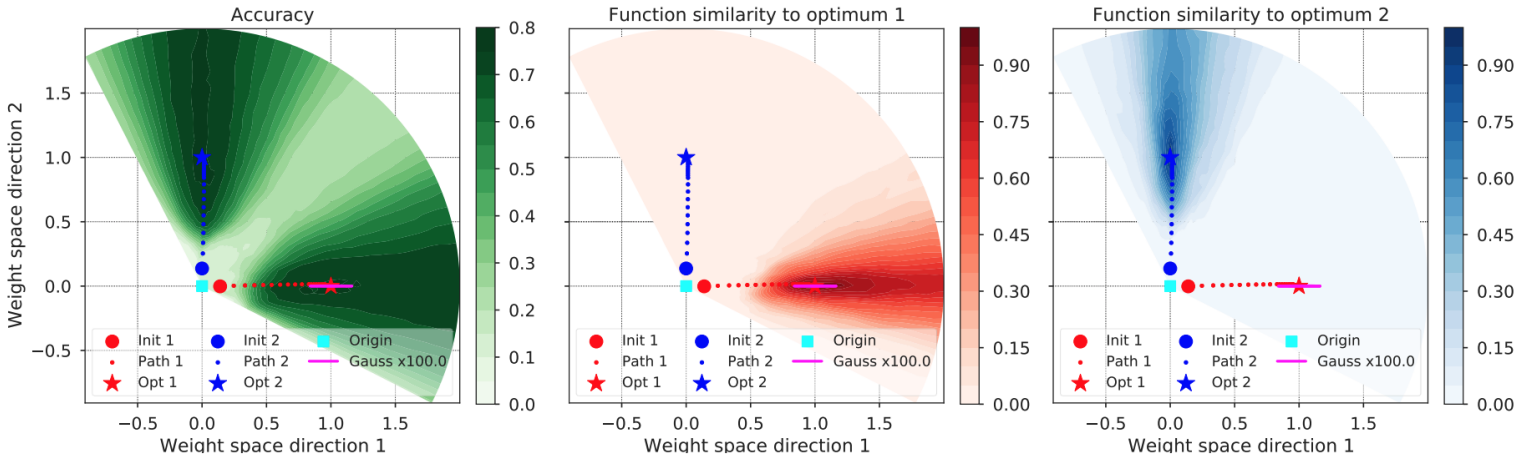


Figure 1: Cartoon illustration of the hypothesis. x-axis indicates parameter values and y-axis plots the negative loss $-L(\boldsymbol{\theta}, \{\boldsymbol{x}_n, y_n\}_{n=1}^{N})$ on train and validation data.



Deep And Classic

Deep And Classic

Deep And Classic

- Ensembles of Deep and Classic Models
- Autoencoders to train efficient feature representations
- Pre-trained models to extract features for efficient ML models
- Interpretable models

Thank You!