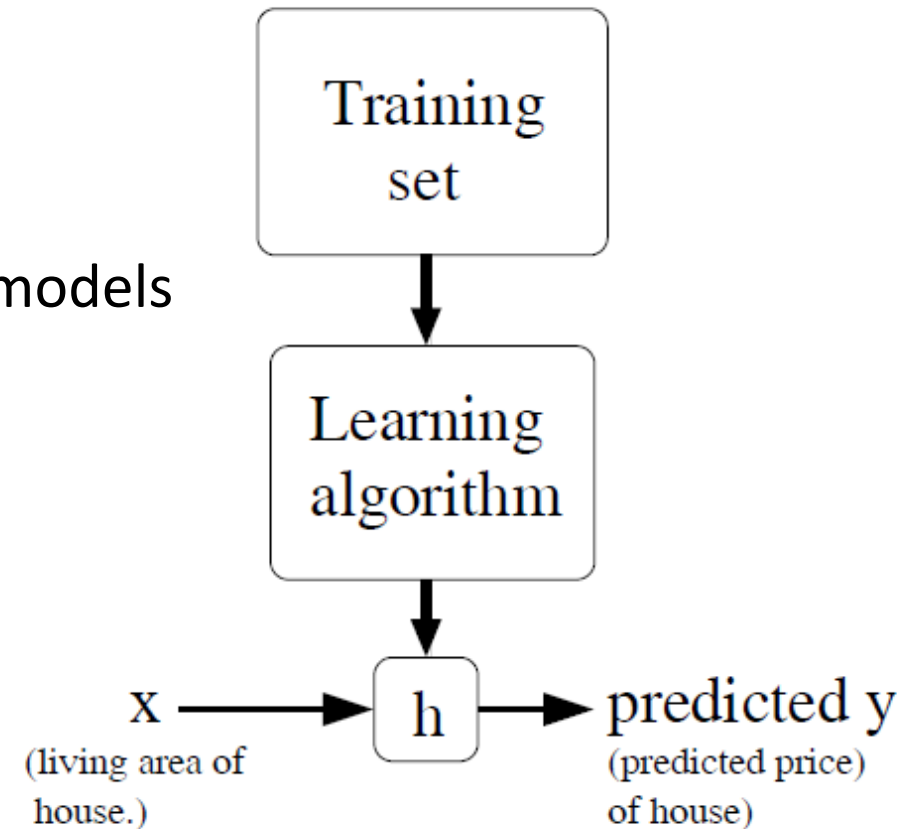


Machine Learning Review

Machine Learning

- ML is about solving problems defined with data
- What typically happens:
 - Define a model family and loss
 - Statistics, modeling of processes, non-parametric models
 - Likelihood, max-margin
 - Search for the best model
 - Numerical or [greedy] combinatorial optimization
 - Apply the model to new data
 - Robustness
 - Regularization



SUPERVISED LEARNING

Data and task

The **data**:

Set of pairs (x, y)

The **task**:

Having x predict y

- Classification: y is a discrete label
- Regression: y is a numerical value

The model

A parametric family of functions f that approximate

$$y \approx f(x, \theta)$$

θ : **parameters** (moving parts set to fit the data), chosen by fitting f to data

To get different f s we set **hyperparameters** (regularization constants, design choices)

SUPERVISED MODELS THAT WE KNOW

Nearest Neighbors

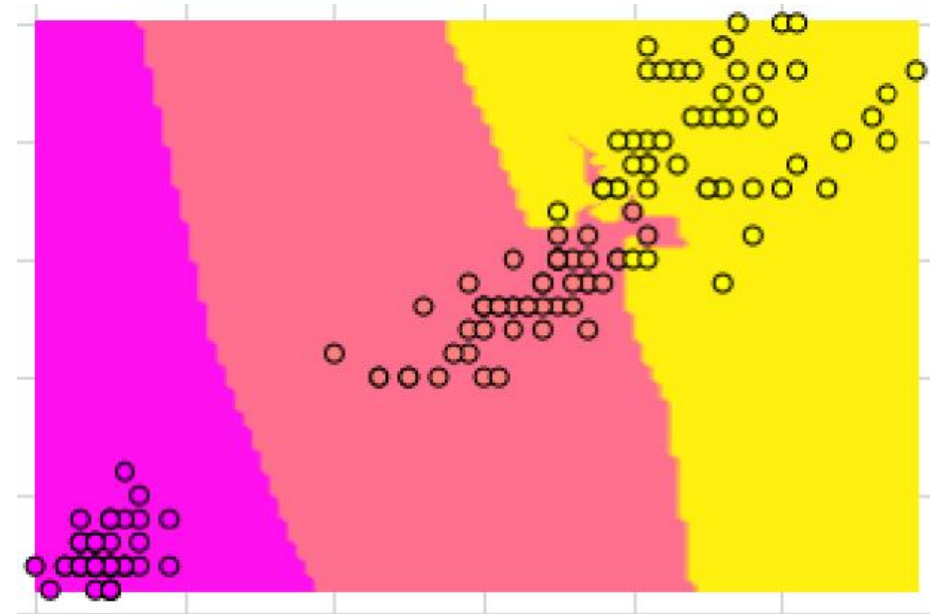
Predict y for a new x based on k nearest neighbors in the dataset.

Parameters:

- None, has to store the whole training set

Hyperparameters:

- k



Linear Regression

$$y \approx f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

Parameters:

- $\theta_0, \theta_1, \theta_2, \dots$
- We set them to:

$$\theta = \arg \min_{\Theta} \frac{1}{2} \sum_i \left(y^{(i)} - f(x^{(i)}) \right)^2 + \frac{\lambda}{2} \sum_j \theta_j^2$$

Hyperparameters:

- λ (larger λ yields small Θ)

Parametric vs Nonparametric Models

Parametric models (e.g., linear regression):

[in statistics]: a family of probability distributions that has a finite number of parameters

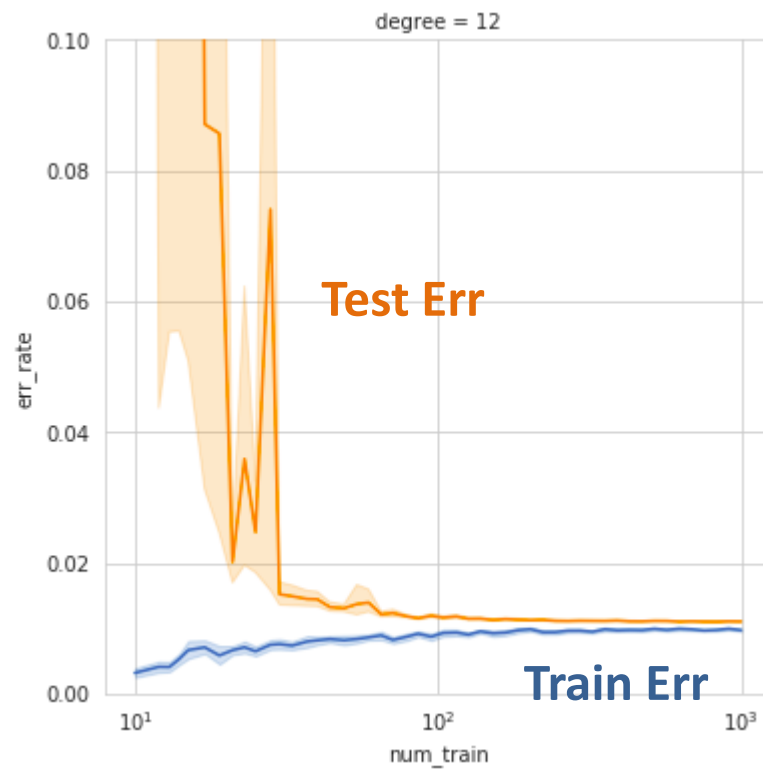
[in ML]: a model whose size DOES NOT grow with amount of data

Non-parametric models (e.g., k-NN):

A model whose size DOES grow with amount of data

Model performance vs dataset size

Parametric



Non-parametric



Naive Bayes: Classification through Generation

y : discrete class labels

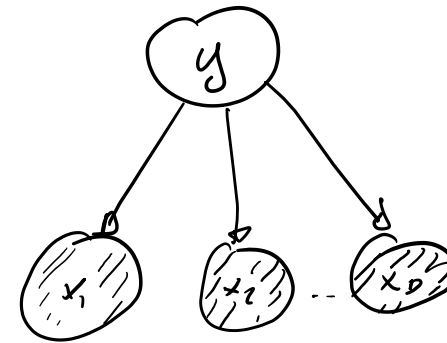
x : typically binary (e.g., presence/absence of a word)

Data generation process:

$$p(y = k) = \pi_k$$

$$p(x|y) \approx \prod_i p(x_i|y)$$

$$p(x_i = 1|y = k) = p_{ik}$$



Parameters:

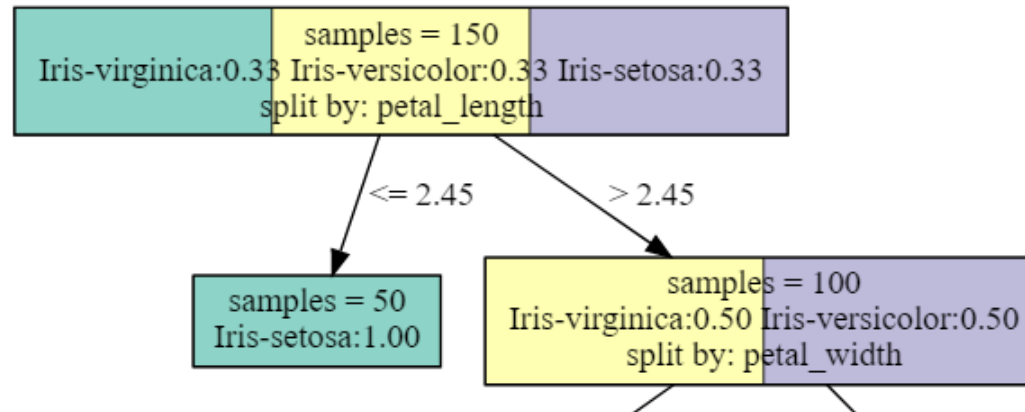
$$\pi_k, p_{ik}$$

Parameter estimation: max likelihood, take observed counts from data

Hyperparameters:

Laplace smoothing (pseudo-counts)

Decision tree



- Each node implements a test
- Subtrees contain fractions of data
- Leaf nodes give classification details
- Parameters: structure, tests, class counts in leaves
- [unpruned] tree grows with data -> nonparametric

Ensembles

Average predictions of many models.

For best results:

- Each model should be strong
- Model errors should be uncorrelated

Note:

strong models are correlated (they agree on samples correctly classified)

Bagging: Bootstrap Aggregation

Decorrelate the models by varying training data

- Draw a bootstrap training sample
- Train the model

Final model simply averages predictions!

Bonus: OOB (Out Of Bag) error estimate:

For each model, record its predictions on the data not in the bootstrap sample. Aggregate the predictions across all models.

Random forest

Average many decision trees.

Core ideas:

1. Bagging: train each tree on a **random** subset of train data (decorrelate trees)
2. Random tree: select each choice from a few **randomly sampled** features (decorrelate trees)
3. Unpruned trees: pruning increases train errs, correlates trees

Random forest

Practical aspects:

- OOB error estimates
- Attribute importance metrics
- Few hyperparams to tune

2nd best all-around classifier after boosted trees

Boosted Trees

Core idea:

Combine many weak (shallow) trees into one strong classifier.

Algorithm loop:

1. Train a tree
2. Reweight the dataset to boost importance of misclassified data
3. Train a new tree
4. Repeat

Finally:

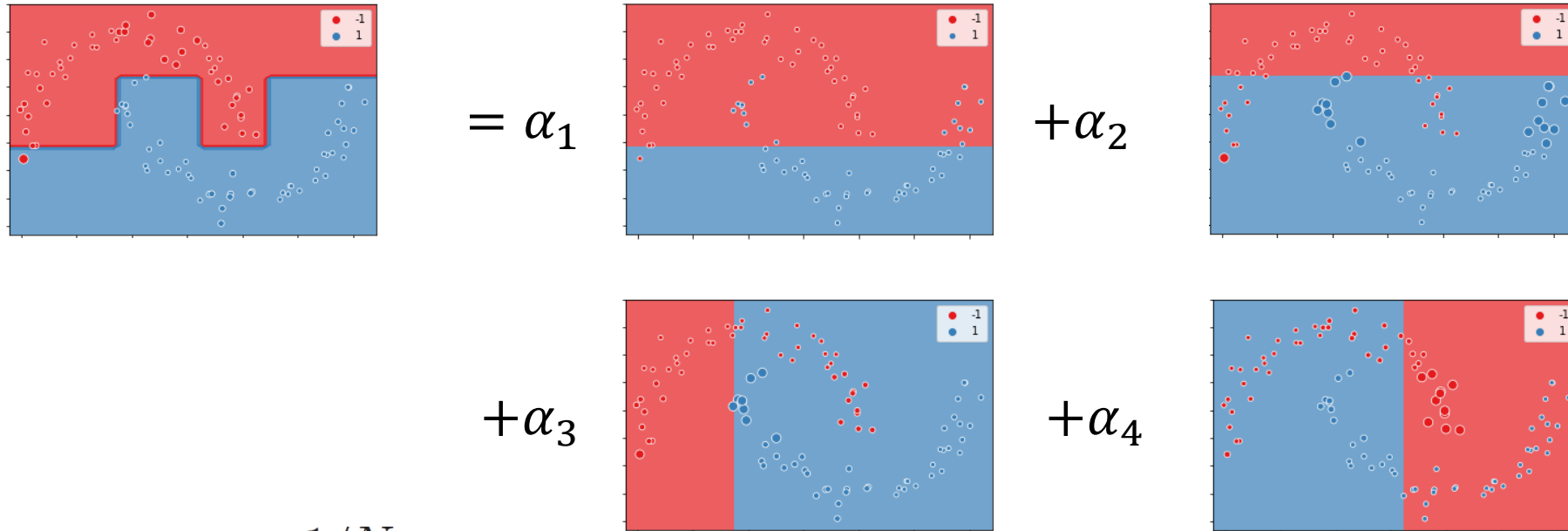
Combine all trees

Note:

Trees are decorrelated.

Each tree fixes errors of previous ones!

Adaboost



$$w_i = 1/N;$$

for $m = 1 : M$ **do**

Fit a classifier $\phi_m(\mathbf{x})$ to the training set using weights \mathbf{w} ;

Compute $\text{err}_m = \frac{\sum_{i=1}^N w_{i,m} \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))}{\sum_{i=1}^N w_{i,m}}$;

Compute $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$;

Set $w_i \leftarrow w_i \exp[\alpha_m \mathbb{I}(\tilde{y}_i \neq \phi_m(\mathbf{x}_i))]$;

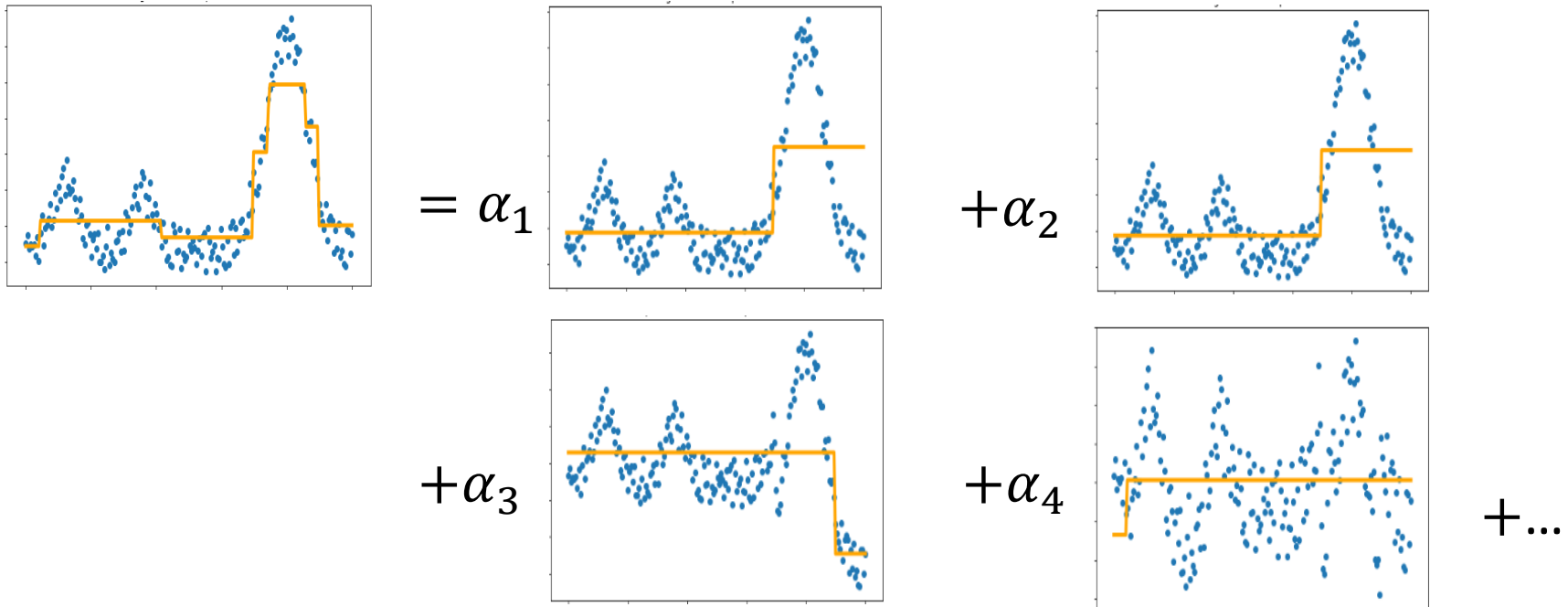
Return $f(\mathbf{x}) = \text{sgn} \left[\sum_{m=1}^M \alpha_m \phi_m(\mathbf{x}) \right]$;

Gradient boosting, XGBoost

Generalize Adaboost to other tasks

$$f(x) = \alpha_1 \phi(x, \gamma_1) + \alpha_2 \phi(x, \gamma_2) + \alpha_3 \phi(x, \gamma_3) + \dots$$

where ϕ is a model with parameters γ_i



GENERALIZATION AND REGULARIZATION

Train Err \neq Test Err

Remembering the training set is easy!

What matters is performance on test set

PAC: Theoretical guarantees on test error rate

Regularization: Preferences on models which decently fit training data. E.g.:

Prefer smaller models – Occam's razor

Prefer models with smoother decision boundaries

PAC: Probably Approximately Correct

Main result:

$$N \geq \frac{1}{\epsilon} \left(\ln \frac{1}{\delta} + \ln |\mathcal{H}| \right)$$

N : train set size

\mathcal{H} : finite hypothesis space

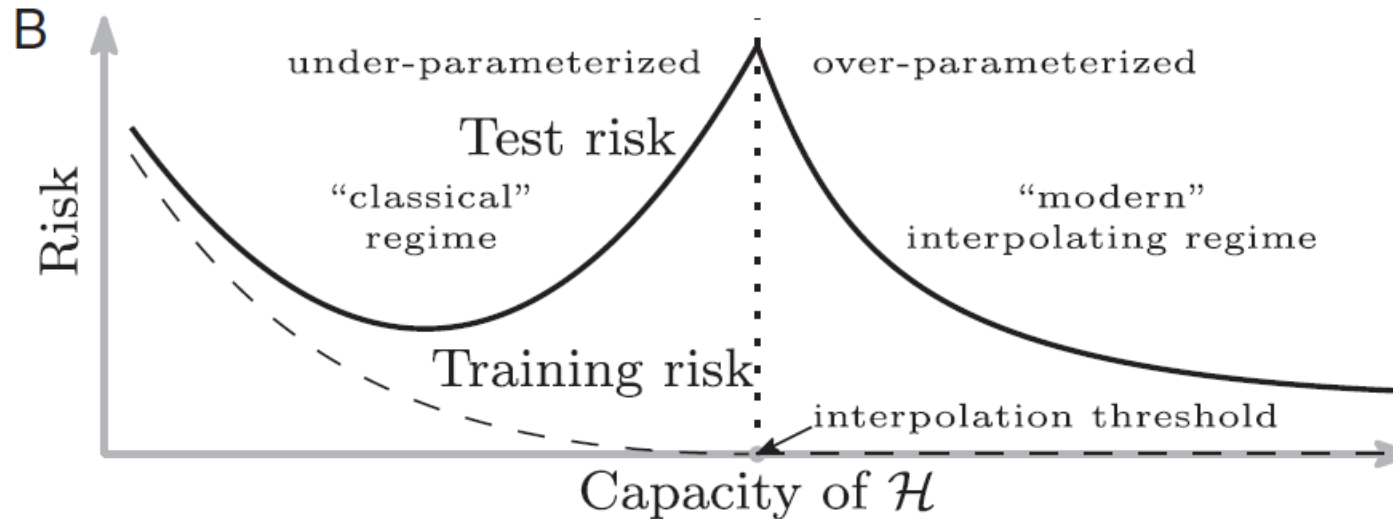
ϵ : desired test err rate

δ : probability that after training *test err* $< \epsilon$

Assumptions: IID data, perfect fit to training set.

Overparameterized models

Double Descent Hypothesis



Near the threshold

Very few models fit the training data,
(think about a polynomial of degree $d+1$
interpolating d points).

Regularization can't help, as there are no
models to choose from 😞

Extremely overparameterized regime

There are many models that fit the training
data!

Regularization can help to choose e.g. a wide-
margin model!

Profit!!!!

Regularization: our modeling preferences

- We need to prefer some hypotheses over others
 - Examples:
 - Linear models are simpler than polynomial
 - Small neural net is simpler than a large one
- Regularization serves to express our preferences about model simplicity
- Typically, we assign a **prior probability** to our models:

$$P(\Theta) = \prod_{i=1}^n \mathcal{N}(\Theta_i; \mu = 0, \sigma = \lambda)$$

Bayesian approach to ML

- What is the model probability after seeing the data \mathcal{D} ?

$$p(\Theta|\mathcal{D}) = \frac{p(\mathcal{D}|\Theta)p(\Theta)}{p(\mathcal{D})}$$

How to make predictions? Integrate over all models:

$$p(y|x, \mathcal{D}) = \int_{\Theta} p(y|x, \Theta)p(\Theta|\mathcal{D})d\Theta$$

Then

$$E[y|x, \mathcal{D}] = \int_y yp(y|x, \mathcal{D})dy$$

But computing $p(y|x, \mathcal{D})$ is often intractable :(

Maximum-a-posteriori

- Instead of integrating over all Θ
- Use the maximally probable Θ :

$$\begin{aligned}\Theta_{MAP} &= \arg \max_{\Theta} p(\Theta | \mathcal{D}) \\ &= \arg \max_{\Theta} \left(\prod_{i=1}^m p(y^{(i)} | x^{(i)}, \Theta) \right) p(\Theta)\end{aligned}$$

- It's like Max. Likelihood with the extra term (regularization).

Linear and Logistic regression

Linear regression

y is continuous

$$p(y|x) = \mathcal{N}(\mu = \Theta^T x, \Sigma)$$

Logistic regression

y is binary

$$\begin{aligned} p(y = 1|x) &= \sigma(\Theta^T x) \\ &= \frac{1}{1 + e^{-\Theta^T x}} \end{aligned}$$

Both

Parameters: Θ

Train to minimize negative log-likelihood with Gaussian

prior $(\log(\mathcal{N}(\Theta; \mu = 0, \sigma^2)) \propto \sum_{j=1}^n (\Theta_j)^2)$

$$\Theta = \arg \min_{\Theta} \sum_i -\log p(y = y^{(i)} | x^{(i)}) + \frac{\lambda}{2} \sum_j \theta_j^2$$

Other priors are possible

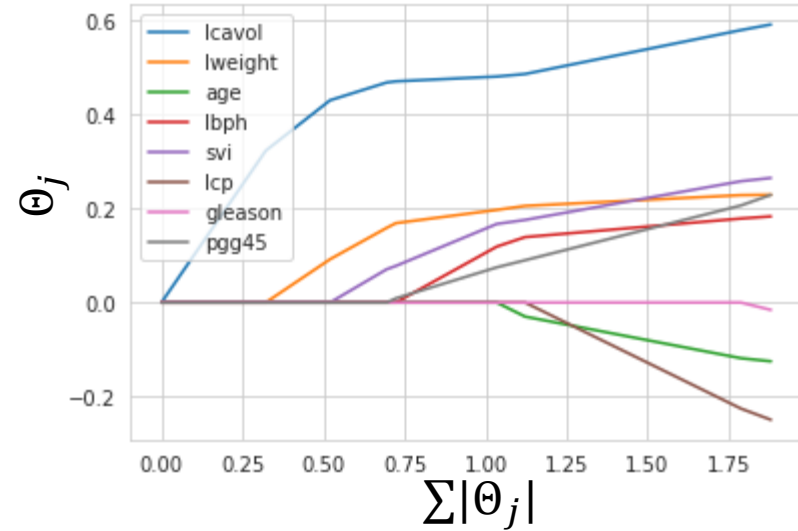
Assume

$$p(\Theta_j) \propto e^{-\lambda|\Theta_j|}$$

Then $-\log p(\Theta_j) = \lambda|\Theta_j|$

This yields LASSO Regression

$$\Theta = \arg \min_{\Theta} \sum_i -\log p(y = y^{(i)} | x^{(i)}) + \lambda \sum_j |\Theta_j|$$

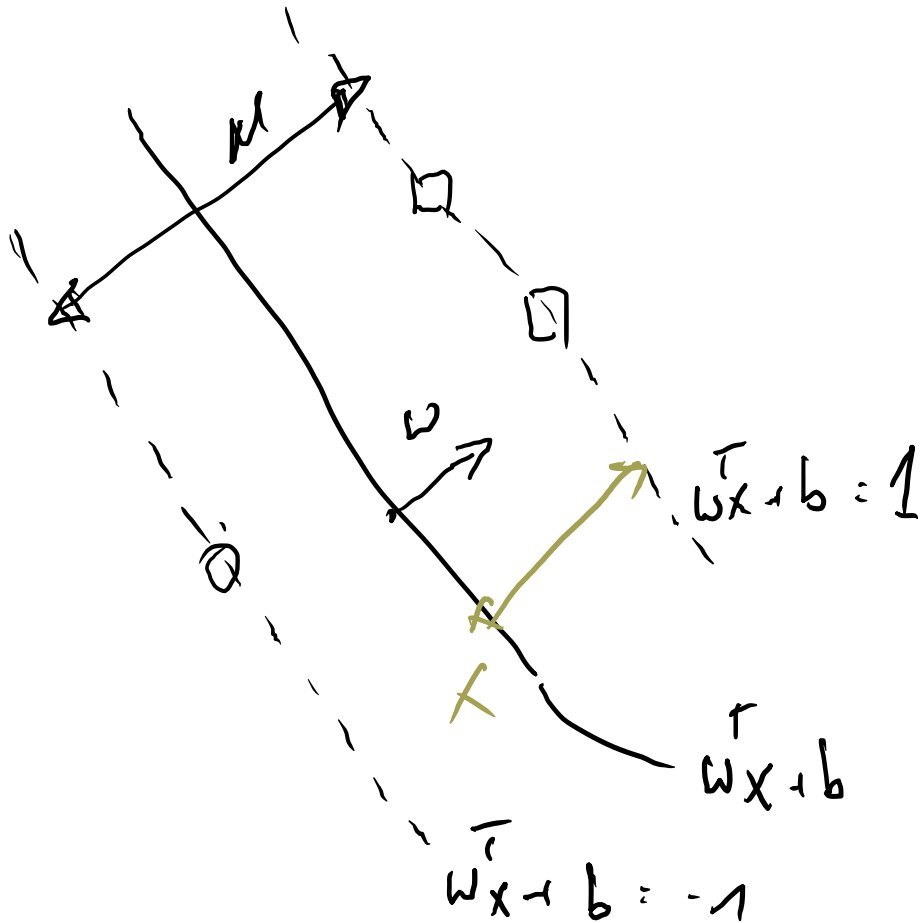


Another view on regularization: Max Margin/SVM

- Task: 2-class classification
- Idea: find a hyperplane yielding max margin



The margin



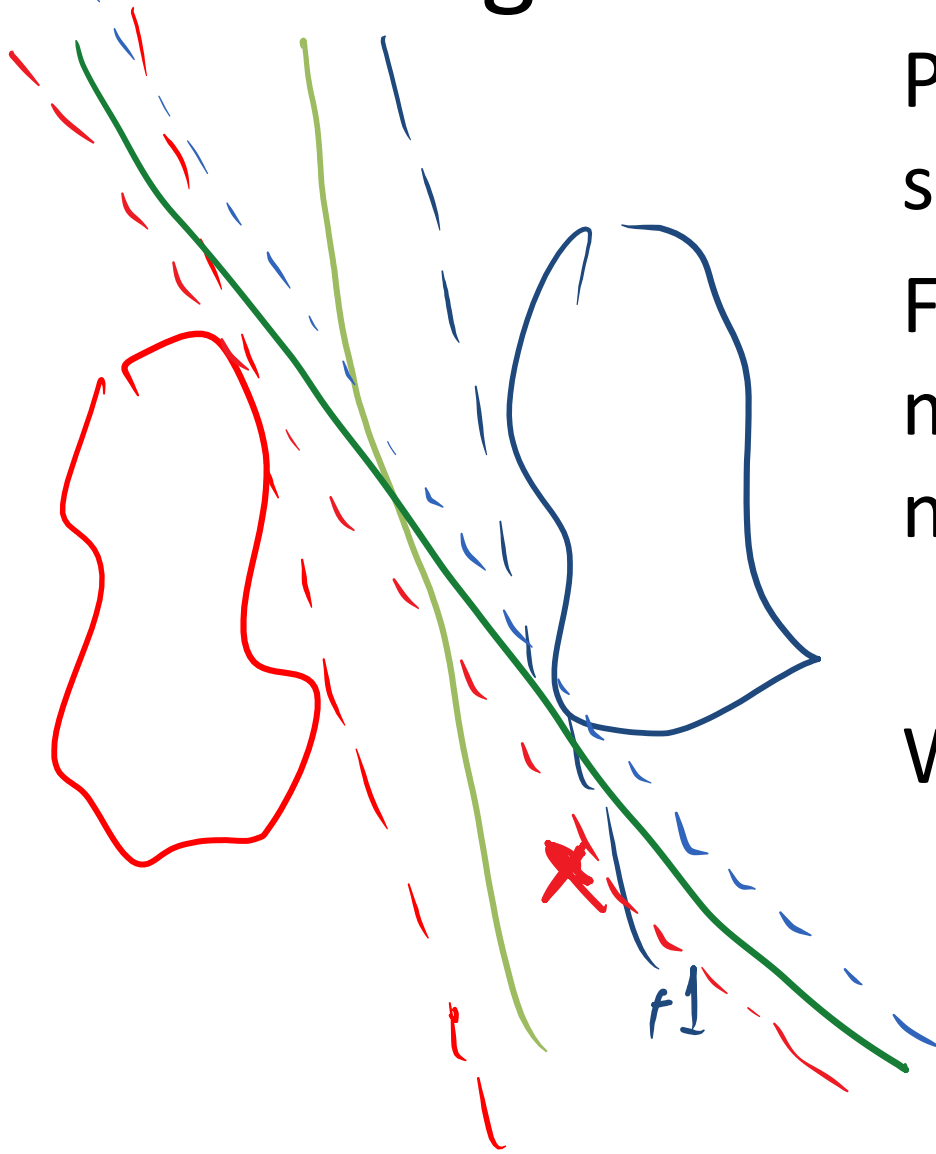
$$\begin{aligned} w^T \left(x + \frac{Mw}{2\|w\|} \right) + b &= \\ &= w^T x + b + \frac{M\|w\|^2}{2\|w\|} = \\ &= \frac{M}{2} \|w\| = 1 \end{aligned}$$

Thus:

$$M = \frac{2}{\|w\|}$$

Maximum margin => minimum weights!

Trading train error for margin



Penalize errors and
samples inside the margin

Find a tradeoff between
margin width and
number of errors!

We want:

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi_i$$

Soft-Margin SVM

The SVM finds weights that minimize

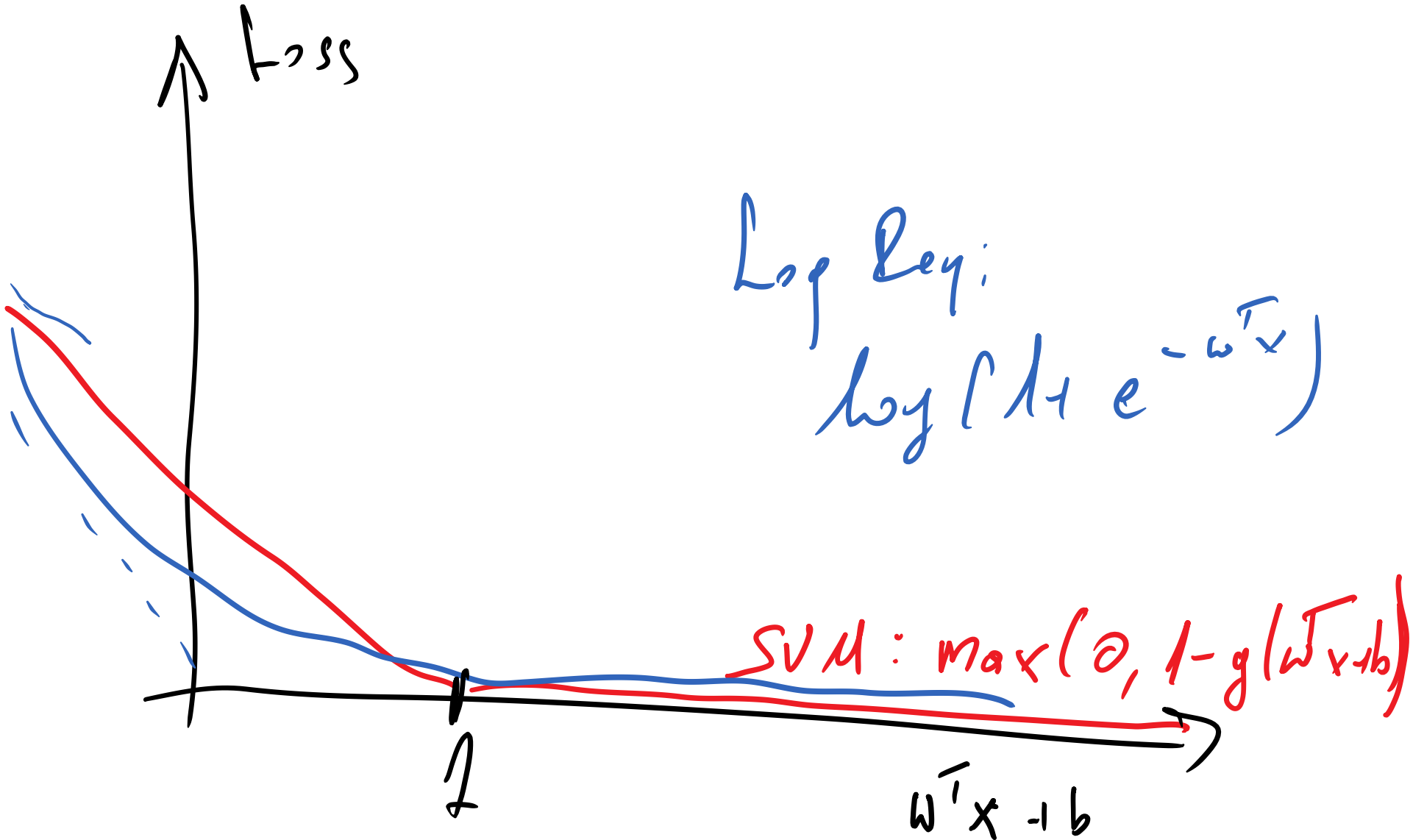
$$\frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_i \xi_i$$

s. t. : $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0 \forall i$

Alternative formulation

$$\frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_i \max \left(0, 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \right)$$

Note: Soft-Margin SVM and LogReg



BEYOND LINEAR MODELS

Idea: transform the data (nonlinearly!)

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

Examples:

- Fixed feature expansion (e.g. polynomial)
- Boosted trees ($\phi(\mathbf{x})$ maps samples to leaves, w has leaf votes)
- Neural networks/deep learning $\phi(\mathbf{x}) = \sigma(\mathbf{w}_H \mathbf{x} + b_H)$

Sometimes all we need are dot-products

Assume

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$$

$$\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}^{(i)})$$

Then

$$f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_i \alpha_i \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}) + b$$

We only need dot-products in the feature $\phi(\cdot)$ space.

Kernels: smart dot-products

Map (nonlinearly) $x \rightarrow \phi(x)$

We only need $\phi(\mathbf{x})^T \phi(\mathbf{y})$

Kernels compute this **in a smart way**:

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$$

K is the **kernel function**.

We never need to compute $\phi(\mathbf{x})$.

This often leads to speedups.

Kernels and feature expansion – further intuitions

Kernels take a **parametric** model and make it **nonparametric**.

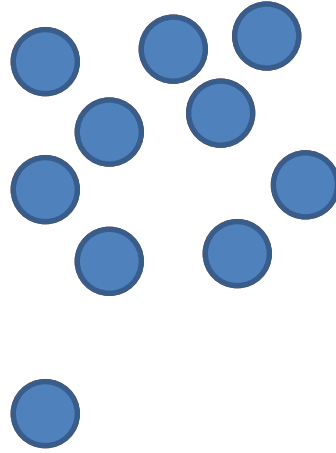
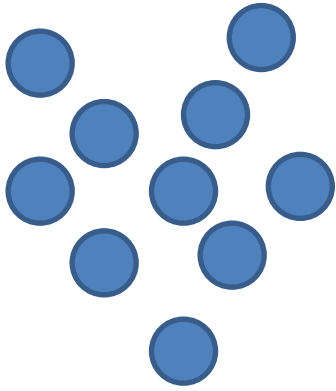
Each data sample has an α_i more data \rightarrow more params.

Kernels can encode our background knowledge – they are like a similarity measure.

Learnable feature expansion (e.g. deep learning) \rightarrow more data may leads to larger models \rightarrow nearly non-parametric

UNSUPERVISED LEARNING

Unsupervised learning



In supervised learning we have labels
In unsupervised we don't have them!

Describe the data!:

- Generate samples similar to the data
- Find clusters (distinct groups of points)
- Reduce the dimensionality
- Find good features that describe the data
- ...

K-Means – a basic algorithm

Input: m input patterns $x^{(i)}$

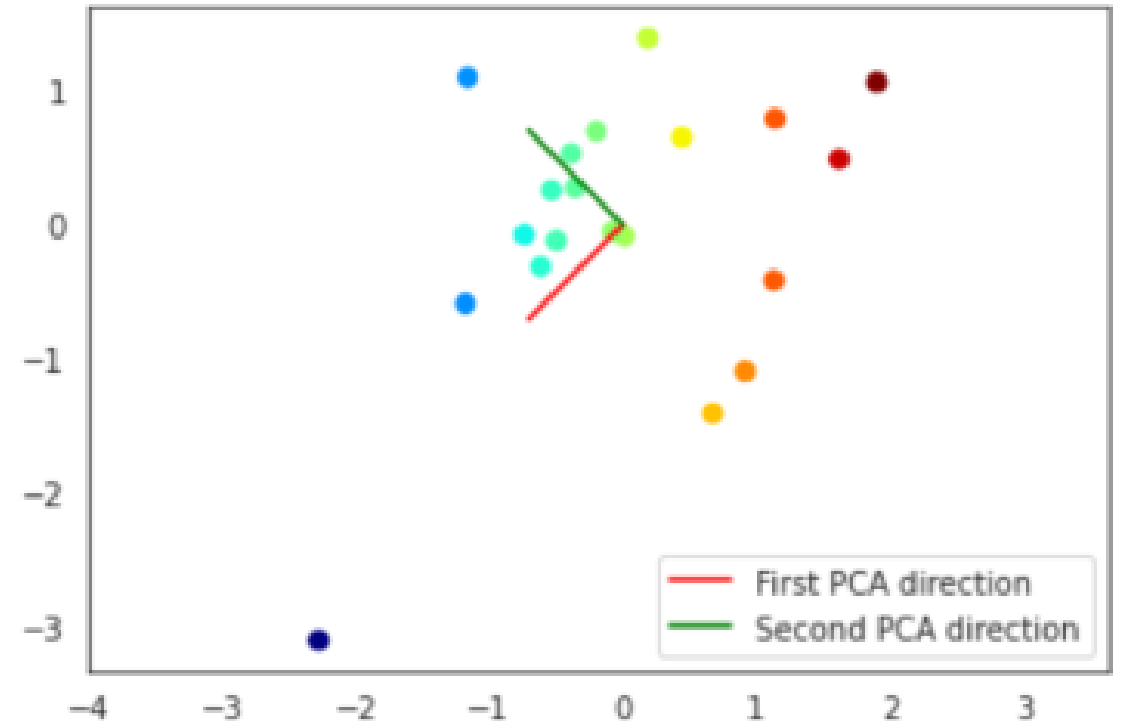
1. Initialize K cluster centers $\mu_1 \dots \mu_K$
randomly, to some input patterns...
even better k-means++: sample data points far away from each other: <http://theory.stanford.edu/~sergei/slides/BATS-Means.pdf>)
2. Loop until convergence:
 1. For all i : set $c^i := \arg \min_j \|x^{(i)} - \mu_j\|^2$
 2. For all j : set $\mu_j := \frac{\sum_i [c^{(i)}=j] x^{(i)}}{\sum_i [c^{(i)}=j]}$

PCA

Find the variation maximizing projection.

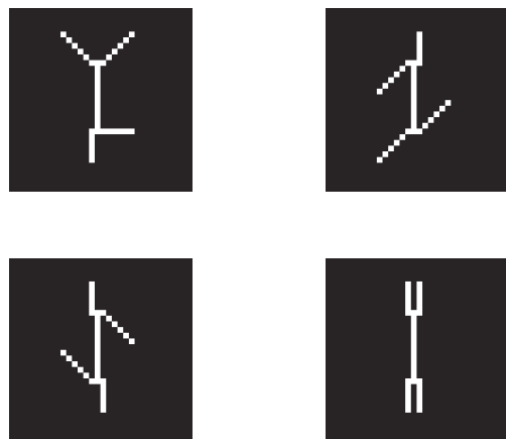
Solution: project onto eigenvectors of data covariance matrix

Linked to SVD decomposition

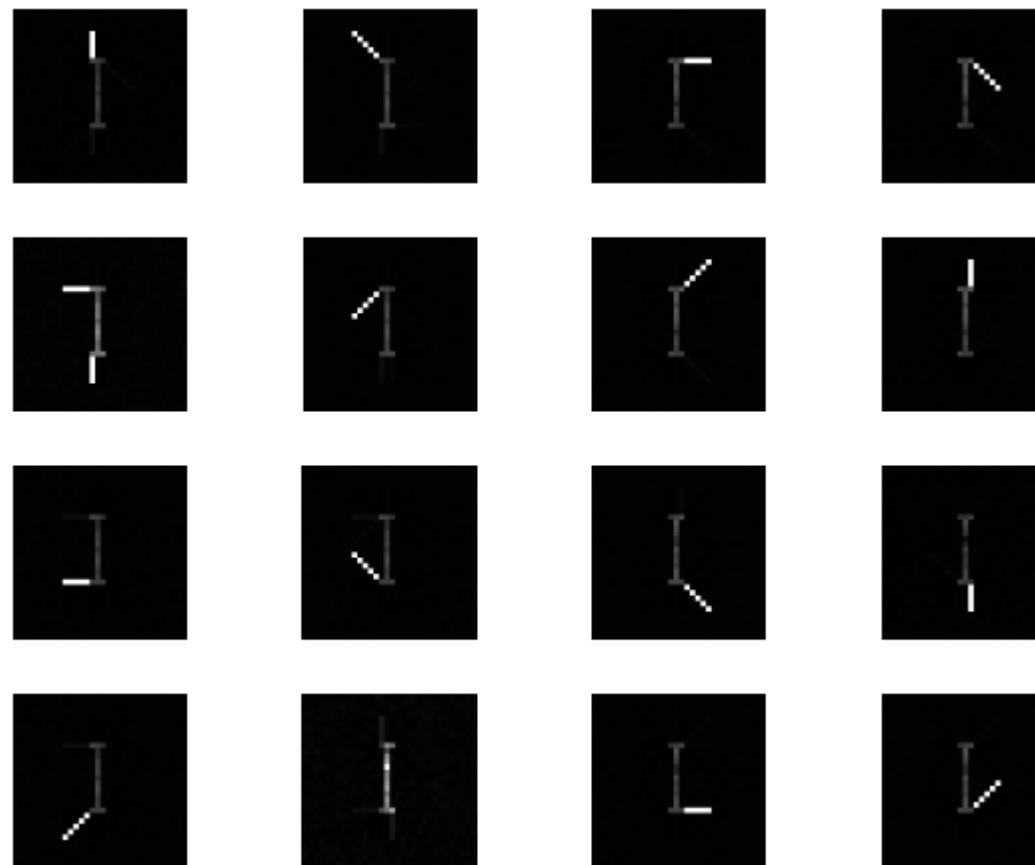


Dictionary learning

Data

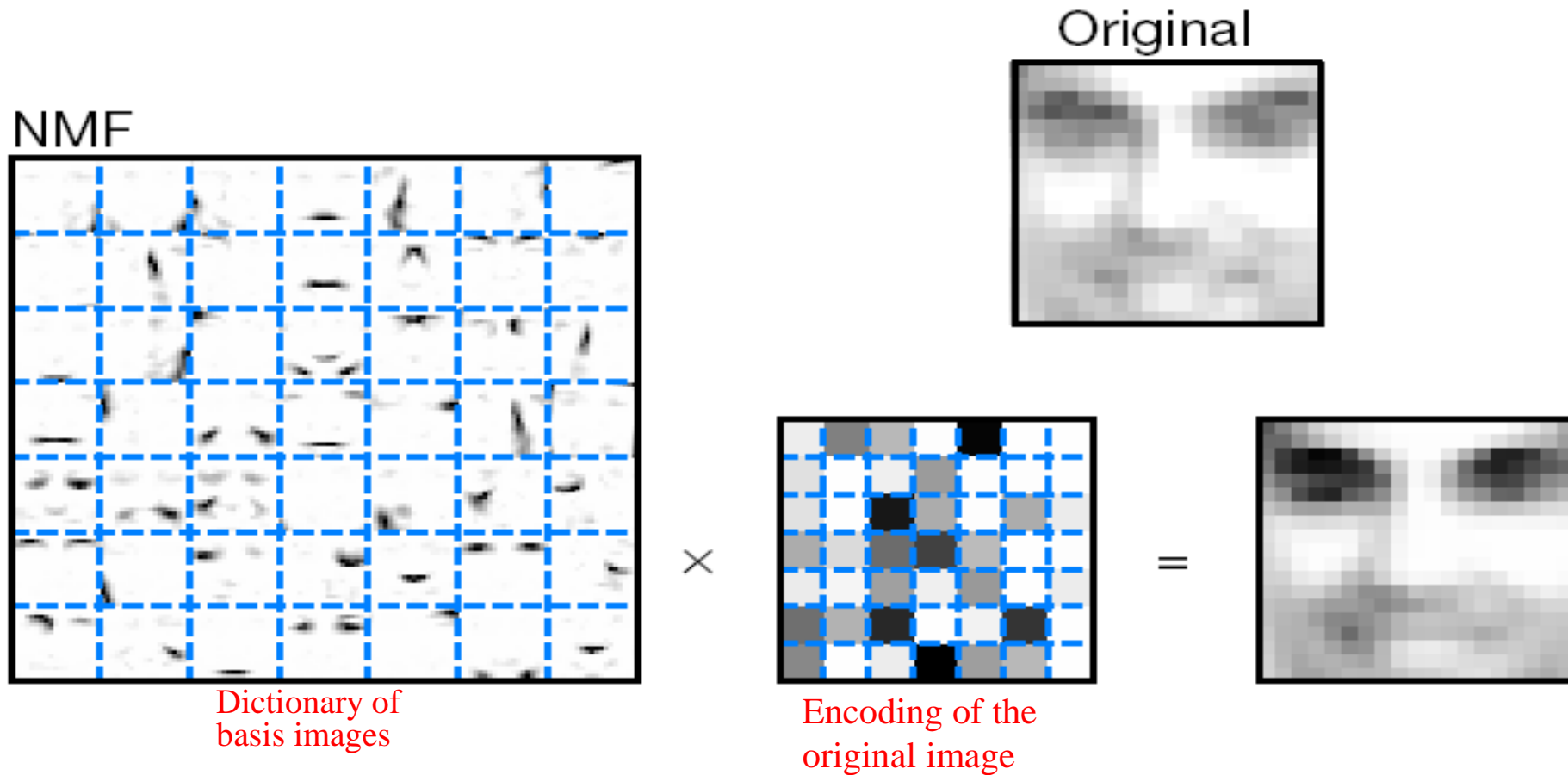


Dictionary



$$x = \mu c$$

Dictionary Learning



Text dictionaries

Small artificial dataset in bag-of-words format

(Topics: ANIMAL d1 and d4, RELIGION d2, FOOD d3)

<u>doc_id</u>	<u>d1</u>	<u>d2</u>	<u>d3</u>	<u>d4</u>
dog	5	1	1	6
bible	0	4	1	0
pizza	1	1	5	2
cat	6	0	1	5
tomato	1	1	6	1
god	0	4	0	0

X =

Data

=

Dictionary

x

Encoding

d1	d2	d3	d4
Dog	dog	dog	dog
Bible	bible	bible	bible
pizza	pizza	pizza	pizza
cat	cat	cat	cat
tomato	tomato	tomato	tomato
god	god	god	god

t1	t2	t3
dog	dog	dog
bible	bible	bible
pizza	pizza	pizza
cat	cat	cat
tomato	tomato	tomato
god	god	god

\times

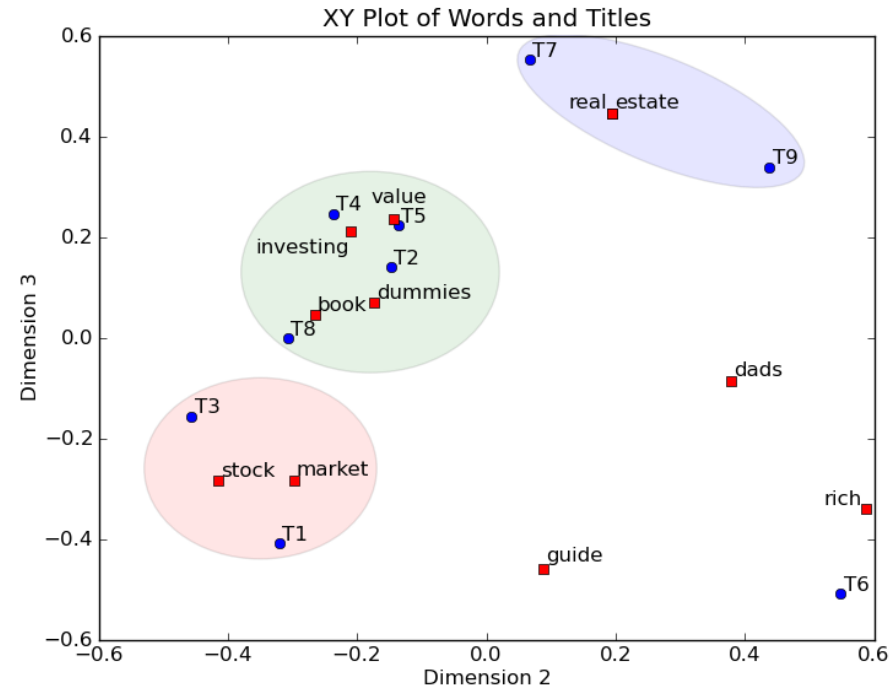
<u>d1</u>	<u>d2</u>	<u>d3</u>	<u>d4</u>

Interpretation:
words for topics

Interpretation:
documents in topics

SVD example

- 9 documents (keywords underlined)
- 1. The Neatest Little Guide to Stock Market Investing
- 2. Investing For Dummies, 4th Edition
- 3. The Little Book of Common Sense Investing: The Only Way to Guarantee You Fair Share of Stock Market Returns
- 4. The Little Book of Value Investing
- 5. Value Investing: From Graham to Buffett and Beyond
- 6. Rich Dad's Guide to Investing: What the Rich Invest in, That the Poor and the Middle Class Do Not!
- 7. Investing in Real Estate, 5th Edition
- 8. Stock Investing For Dummies
- 9. Rich Dad's Advisors: The ABC's of Real Estate Investing: The Secrets of Finding Hidden Profits Most Investors Miss



PRACTICAL ASPECTS OF LEARNING

What's the goal?

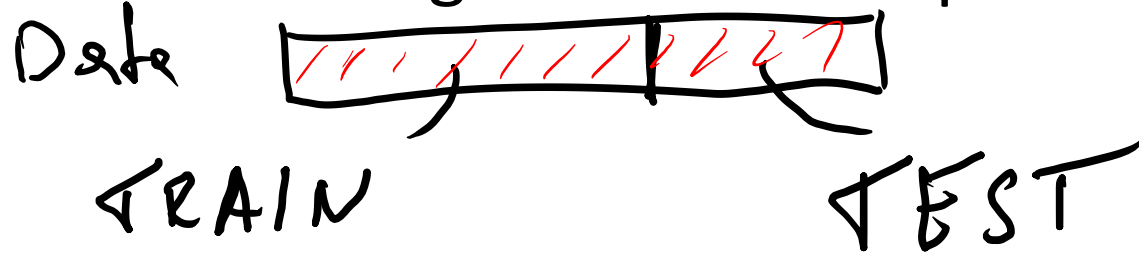
The **ultimate** goal of learning:
do well on **new/unseen** data

ML projects are defined by
metrics and **test sets**.

Honest estimates: Hold-out set

Large data case!!!

- Split the training data into two parts:



- Train only on training, then test on testing.
- Often we do a three-way split:

- Diagram illustrating a three-way split of training data:
-
- The diagram shows a horizontal bar representing a dataset. Two vertical lines divide the bar into three parts. The first part is labeled 'TRAIN', the second part is labeled 'VALIDATION', and the third part is labeled 'TEST'.
- Then:
 - Train many models on training (different algos, parameters)
 - Use validation to choose best model
 - Test on testing

Cross-validation

Small data case!!

- Hold-out set makes inefficient data use
- Idea:

- Divide the data into k sets ($\sim 5, 10$)

For $i=1..k$

Train on all but the i -th set

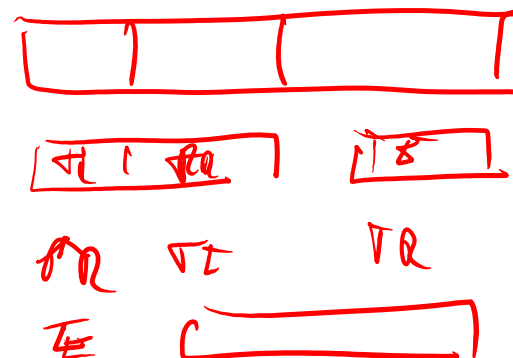
may further split to choose the model...

Test on the i -th set

Finally:

take the answers on the testing sets and use them to compute the performance measures

- Extreme case: leave-one-out (jackknife) – always use all but one sample to train!

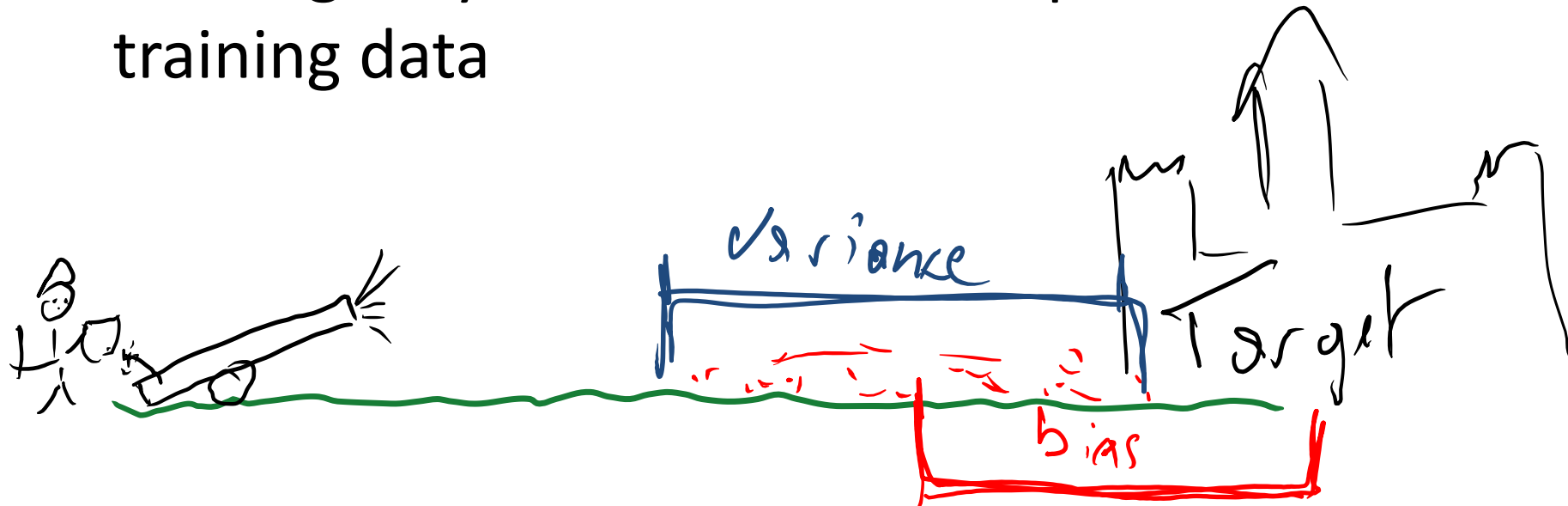


Bootstrap

- Small data case!!
- Sample with replacement m samples
 - About 37% will not be selected
- Train on the selected samples
- Test on the remaining ones
- Optionally repeat.

Bias-Variance: two sources of error!

- The **bias** captures how well our family of functions (hypothesis space) matches the data.
- The **variance** captures how the results of training vary with different samples from the training data



How to lower the bias?

- Choose more powerful/better models:
 - Understand the data and choose a matching model
 - Describe the data with more attributes
 - Expand the data, e.g.:
$$x \rightarrow [1, x, x^2, x^3, \dots]$$
- This usually increases the Hypothesis space


How to lower variance?

- Get more data (or generate synthetic, e.g. rotate and shear pictures)
- Select only the most important inputs
- **Constrain the models:**
 - Simpler models
 - Regularize the models:
Assign a probability distribution to the models and choose the most probable ones
- **Average the models**
 - Very powerful
 - Also called “ensemble learning”, boosting, bagging
 - Requires that the models make uncorrelated errors

Approximations we take

- We want accuracy on UNKNOWN TEST DATA
- Approximation: Cross-Validation, hold-out set
- Can't directly optimize acc (non-differentiable, NP-hard...)
- Proxy: optimize a loss function
- Often impossible exactly –use some greedy algo

Errors can come at all stages

- Data:
 - Is it representative of the problem
 - Does it cover all possible variations (e.g. in France “z” is )
 - Can you get more of it? Generate? Transform?
- Prior beliefs:
 - Does the architecture you choose match the problem?
 - Maybe you know something (e.g. invariants, predominating probability distribution...)
- Loss function:
 - Does it make sense? Is it for classification/regression? Do smaller loss correspond to better performance?
- Training algorithm:
 - Do you reach the minimum of what you optimize?
 - Intentionally? How about early stopping?
- Performance measures:
 - do you separate train from test data?
 - How do train and test errors compare?

Example

Logistic regression classifier makes 10% errors

SVM with Gaussian Kernel makes 20% 😞

- Use the same loss – take linear and nonlinear SVM, which one has the lowest loss?

(don't change training, just loss computation):

- Linear classifier -> is the nonlinear SVM correct??
 - Maybe it is too regularized?
- Nonlinear -> how is your train and test error, do you over-fit?
 - Do you use regularization? Can you increase it?
 - Can you get more training data?
 - Maybe the linear classifier is also over-fitting?