Recognition

Image classification

- Given an image, produce a label
- Label can be:
  - 0/1 or yes/no: *Binary classification*
  - one-of-k: *Multiclass classification*
  - 0/1 for each of k concepts: *Multilabel classification*

Image classification - Binary classification

Is this a dog?

Yes

Image classification - Multiclass classification

Which of these is it:
dog, cat or zebra?
Dog

Image classification - Multilabel classification

Is this a dog? Yes
Is this furry? Yes
Is this sitting down? Yes

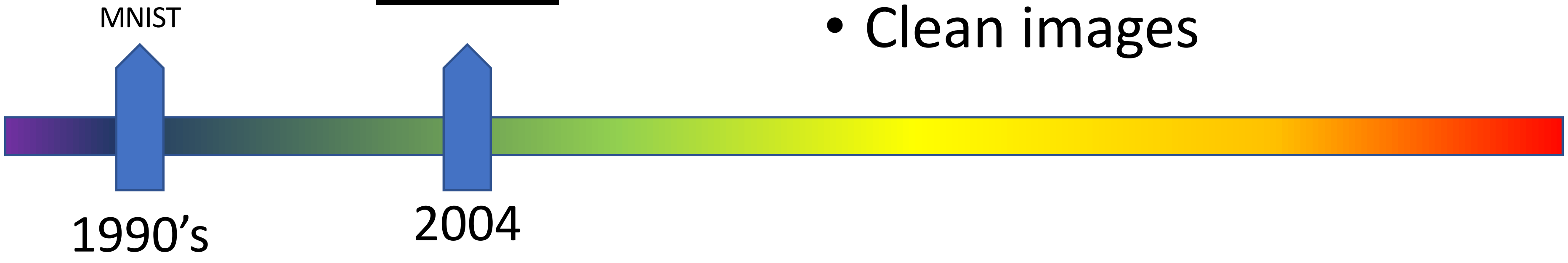
A history of classification : MNIST

- 2D
- 10 classes
- 6000 examples per class

1990’s

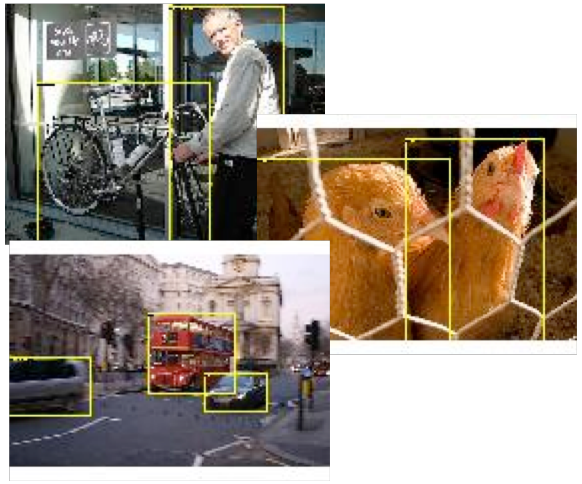
A history of classification : Caltech 101

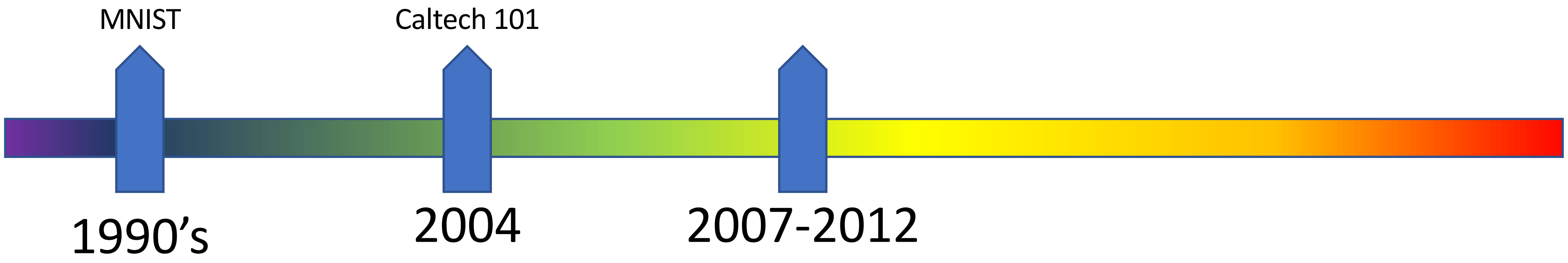
- 101 classes
- 10 classes
- 30 examples per class
- Strong category-specific biases
- Clean images



A history of classification: PASCAL VOC

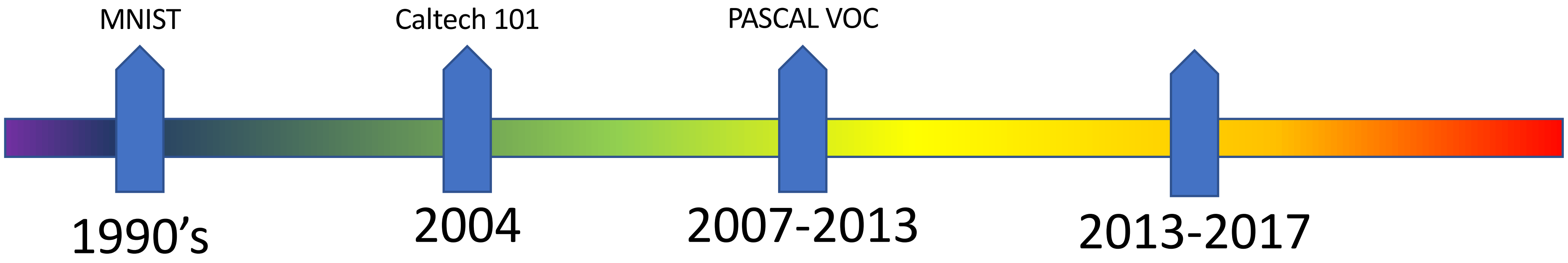
- 20 classes
- ~500 examples per class
- Clutter, occlusion, natural scenes



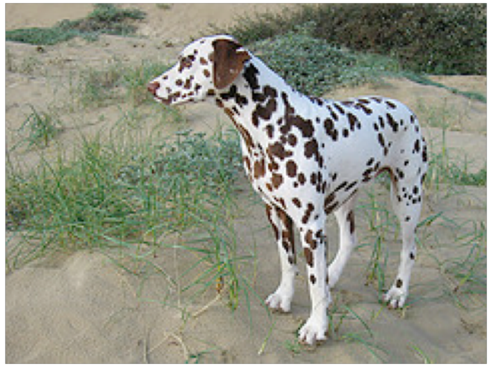


A history of classification: ImageNet

- 1000 classes
- ~1000 examples per class
- Mix of cluttered and clean images

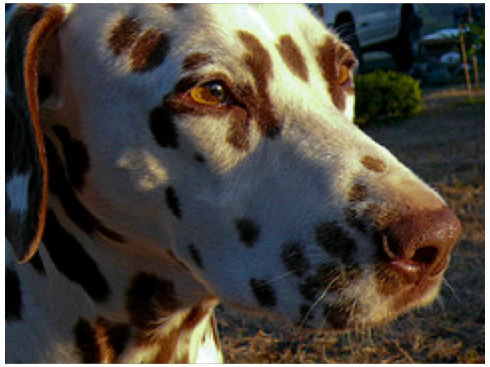


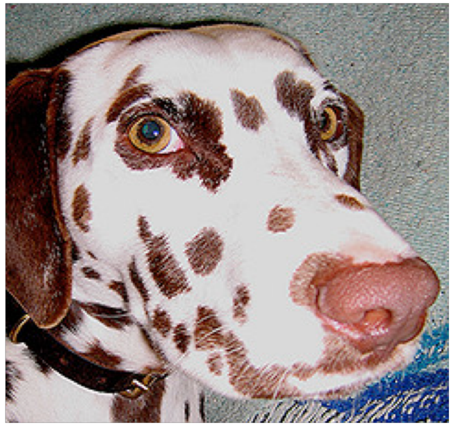
Why is recognition hard?



Pose variation

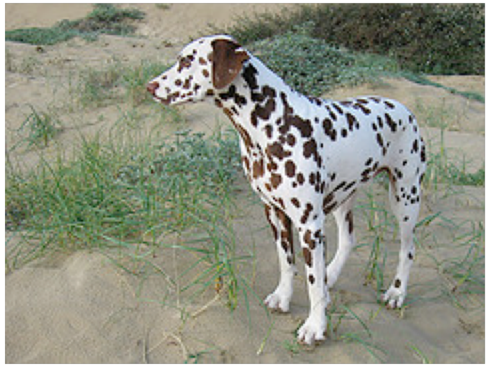
Why is recognition hard?





Lighting variation

Why is recognition hard?

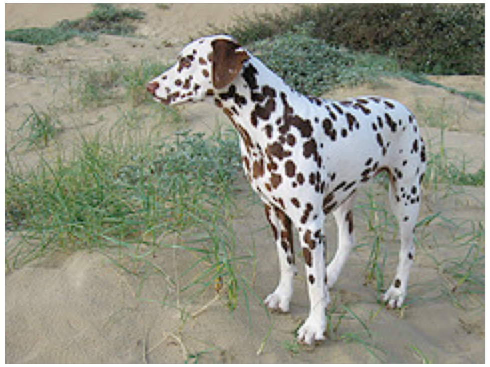


Scale variation

Why is recognition hard?

Clutter and occlusion

Why is recognition hard?



Intrinsic intra-class variation

Why is recognition hard?

Inter-class similarity

The language of recognition

- Boundaries of classes are often fuzzy
- “A dog is an animal with four legs, a tail and a snout”
- Really?

The language of recognition

- "… Practically anything can happen in an image and furthermore practically everything did" - Marr
- Much better to talk in terms of *probabilities*

$$\mathcal{X} \text{ :Images}$$
$$\mathcal{Y} \text{ :Labels}$$
$$\mathcal{D} \text{ :Distribution over } \mathcal{X} \times \mathcal{Y}$$

- *Joint distribution of images and labels* : P(x,y)
- *Conditional distribution of labels given image* : P(y|x)

Learning

- We are interested in the conditional distribution $P(y|x)$
- Key idea: teach computer visual concepts by *providing examples*

$\mathcal{X}$:Images

$\mathcal{Y}$:Labels

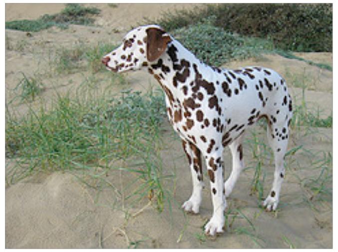
$\mathcal{D}$:Distribution over $\mathcal{X} \times \mathcal{Y}$

Training Set

$$S = \{(x_i, y_i) \sim \mathcal{D}, i = 1, \dots, n\}$$

Example

- Binary classifier “Dog” or ”not Dog”
- Labels: {0, 1}
- Training set

{( , 1), ( , 1), ( , 0) , … }

Choosing a model class

- Will try and find P(y = 1 | x)
- P(y=0 | x) = 1 - P(y=1 | x)
- Need to find $h : \mathcal{X} \rightarrow [0, 1]$
- But: *enormous number of possible mappings*

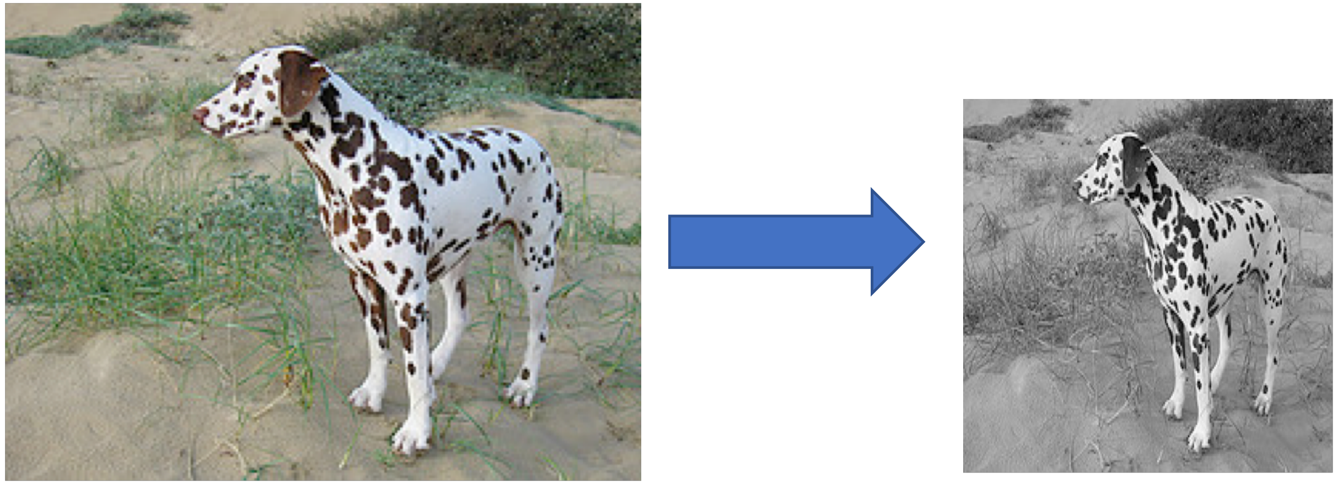
Choosing a model class

$$h : \mathcal{X} \rightarrow [0, 1]$$

- Assume h is a linear classifier in feature space
- Feature space?
- Linear classifier?

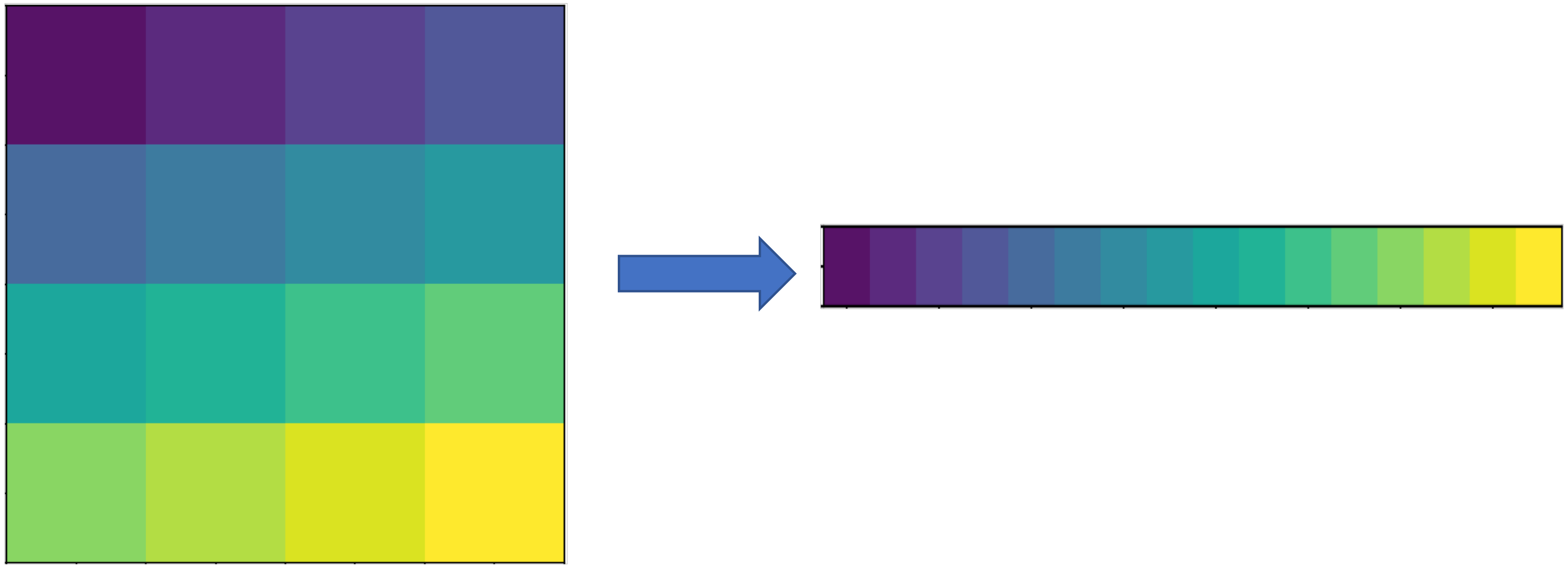
Feature space: representing images as vectors

- Represent an image as a vector in $\mathbb{R}^d$
- Simple way: step 1: convert image to gray-scale and resize to fixed size



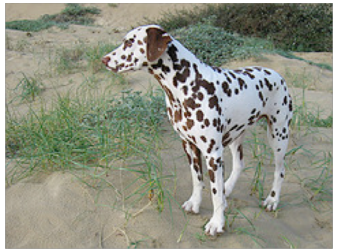
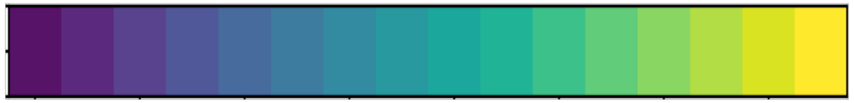
Feature space: representing images as vectors

- Step 2: Flatten 2D array into 1D vector



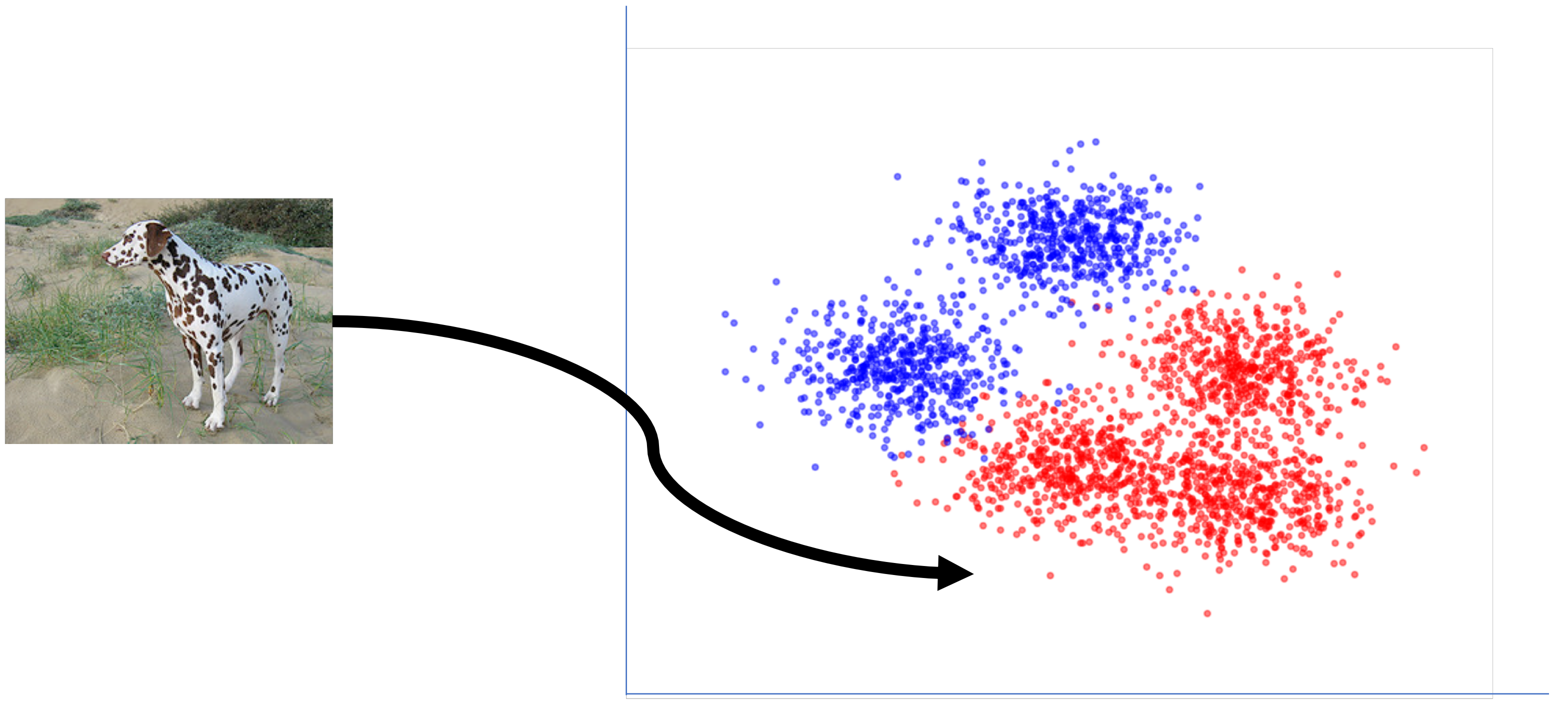
Feature space: representing images as vectors

- Can represent this as a *function* that takes an image and converts into a vector

$\phi($  $)$ = 

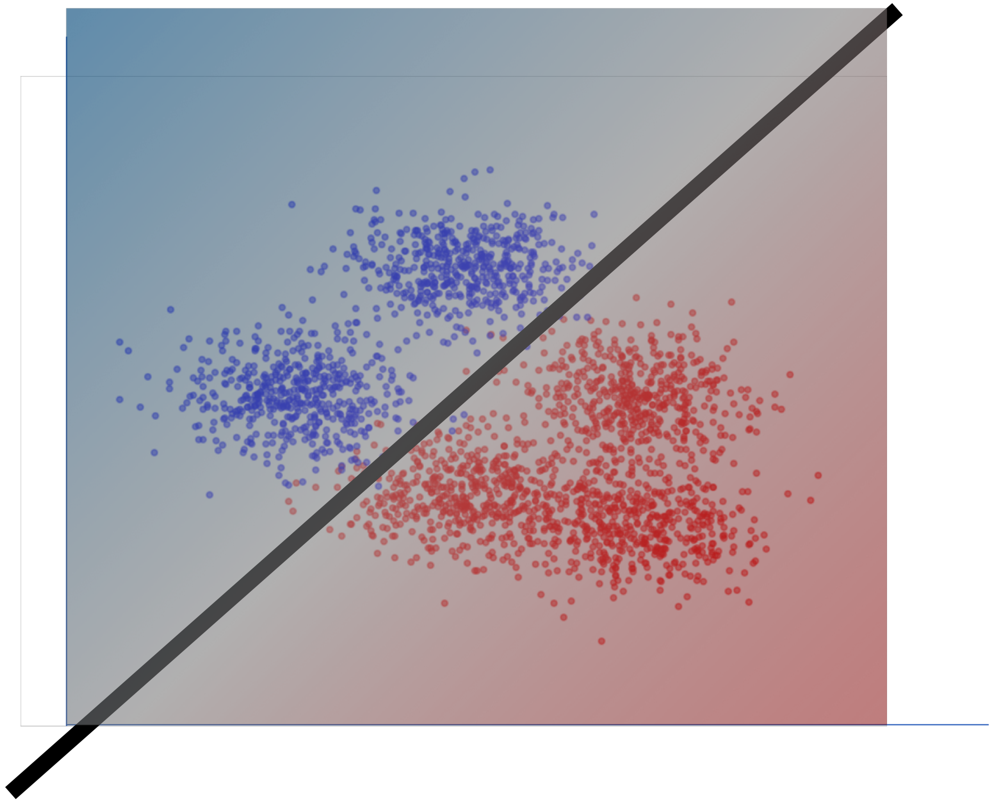
Linear classifiers

- Given an image, can use ϕ to get a vector and plot it as a point in high dimensional space



Linear classifiers

- A linear classifier corresponds to a hyperplane
  - Equivalent of a line in high-dimensional space
  - Equation: $w^T x + b = 0$
- Points on the same side are the same class

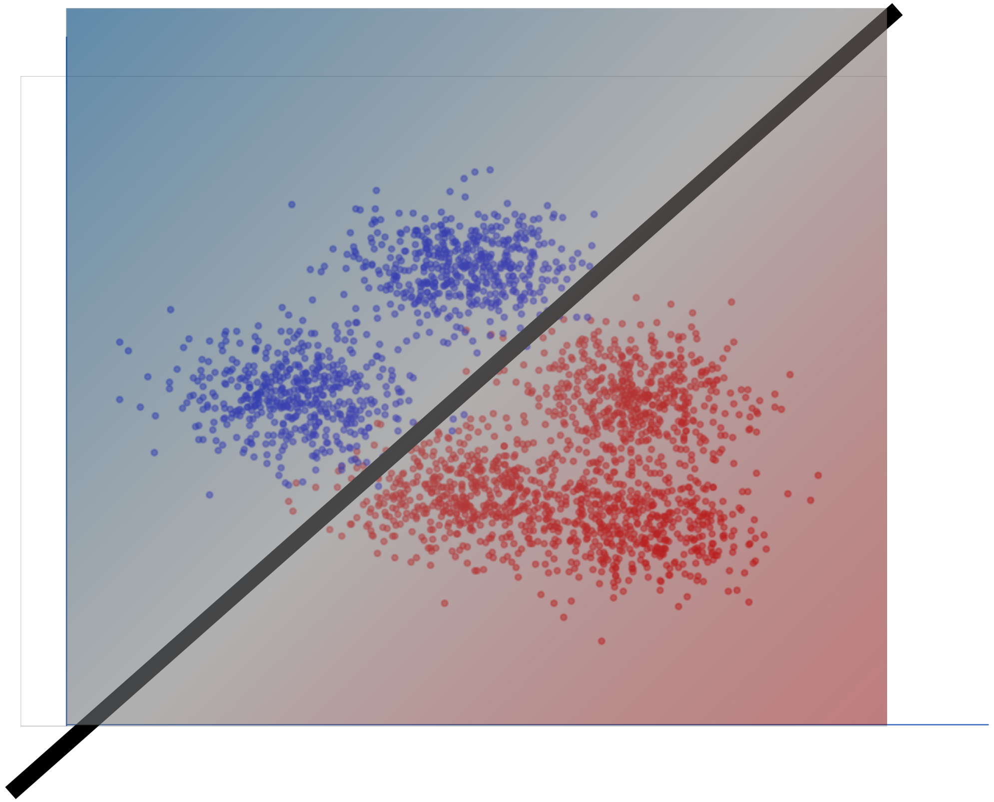


Linear classifiers

- p(y = 1 | x) is high on the red side and low on the blue side
- A common way of defining p:

$$p(y = 1\,|x) = \sigma(w^T x + b)$$

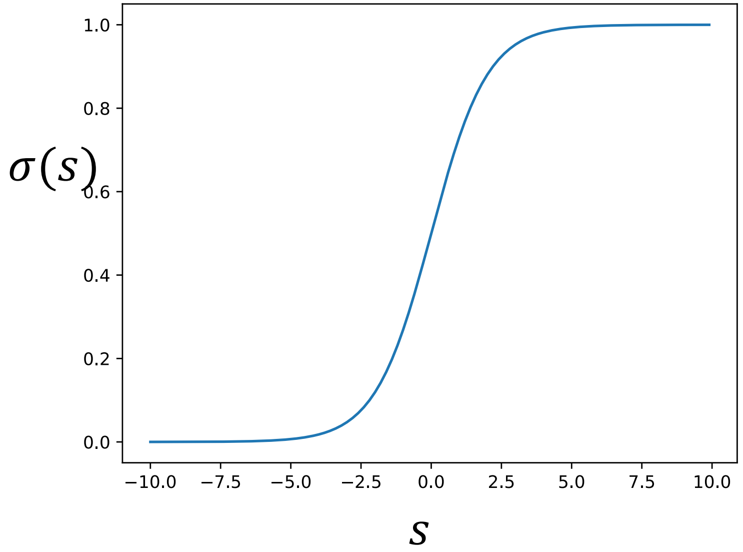
$$= \underbrace{\frac{1}{1 + e^{-(w^T x + b)}}}_{\text{sigmoid function}}$$



Linear classifiers in feature space

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

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



Linear classifiers in feature space

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

- *Family* of functions depending on **w** and b
- Each function is called a *hypothesis*
- Family is called a *hypothesis class*
- Hypotheses indexed by **w** and b
- Need to find the best hypothesis = need to find best **w** and b
- **w** and b are called *parameters*

Training: Choosing the best hypothesis

- Use training set to find *best-fitting* hypothesis

$$S = \{(x_i, y_i) : i = 1, \ldots, n\}$$

- Question: how do we define fit?

Training: Choosing the best hypothesis

- Use training set to find *best-fitting* hypothesis
- Question: how do we define fit?
- Given (x,y), and candidate hypothesis $h(\cdot; \mathbf{w}, b)$
  - $h(x; \mathbf{w}, b)$ is estimated probability label is 1
  - Idea: compute estimated probability for true label y
  - Want this probability to be high
  - *Likelihood*

$$li(h(x; \mathbf{w}, b), y) = \begin{cases} h(x; \mathbf{w}, b) & \text{if } y = 1 \\ 1 - h(x; \mathbf{w}, b) & \text{ow} \end{cases}$$

An alternate expression for the hypothesis

$$li(h(x;\mathbf{w},b),y) = \begin{cases} h(x;\mathbf{w},b) & \text{if } y = 1 \\ 1 - h(x;\mathbf{w},b) & \text{ow} \end{cases}$$

An alternate expression for the hypothesis

$$li(h(x;\mathbf{w},b),y) = \begin{cases} h(x;\mathbf{w},b) & \text{if } y=1 \\ 1-h(x;\mathbf{w},b) & \text{ow} \end{cases}$$

$$li(h(x;\mathbf{w},b),y) = h(x;\mathbf{w},b)^{y}(1-h(x;\mathbf{w},b))^{(1-y)}$$

Training: Choosing the best hypothesis

$$li(h(x;\mathbf{w},b),y) = h(x;\mathbf{w},b)^y(1-h(x;\mathbf{w},b))^{(1-y)}$$

- Likelihood of a single data point
- Fit = *total likelihood of entire training dataset*

$$S = \{(x_i, y_i) \sim \mathcal{D}, i = 1, \ldots, n\}$$

$$\prod_{i=1}^{n} h(x_i;\mathbf{w},b)^{y_i}(1-h(x_i;\mathbf{w},b))^{(1-y_i)}$$

Training: Choosing the best hypothesis

$$\prod_{i=1}^{n} h(x_i; \mathbf{w}, b)^{y_i} (1 - h(x_i; \mathbf{w}, b))^{(1-y_i)}$$

- Use log likelihood

$$lli(\mathbf{w}, b) = \sum_{i=1}^{n} y_i \log h(x_i; \mathbf{w}, b) + (1 - y_i) \log(1 - h(x_i; \mathbf{w}, b))$$

- Pick the hypothesis that maximizes log likelihood
  - Each hypothesis corresponds to a setting of **w** and b
  - *Maximization problem*

$$\max_{\mathbf{w}, b} \sum_{i=1}^{n} y_i \log h(x_i; \mathbf{w}, b) + (1 - y_i) \log(1 - h(x_i; \mathbf{w}, b))$$

Training: Choosing the best hypothesis

- Maximizing log likelihood = *Minimizing average negative log likelihood*

$$\max_{\mathbf{w},b} \sum_{i=1}^{n} y_i \log h(x_i;\mathbf{w},b) + (1-y_i)\log(1-h(x_i;\mathbf{w},b))$$

$$\equiv \min_{\mathbf{w},b} -(\sum_{i=1}^{n} y_i \log h(x_i;\mathbf{w},b) + (1-y_i)\log(1-h(x_i;\mathbf{w},b)))$$

$$\equiv \min_{\mathbf{w},b} \frac{-1}{n}(\sum_{i=1}^{n} y_i \log(h(x_i;\mathbf{w},b)) + (1-y_i)\log(1-h(x_i;\mathbf{w},b)))$$

Training: Choosing the best hypothesis

- Negative log likelihood is a *loss function*

$$L(h(x;\mathbf{w},b),y) = -y\log h(x;\mathbf{w},b) + (1-y)\log(1-h(x;\mathbf{w},b))$$

- *Training = minimizing average loss on a training set*

$$\min_{\mathbf{w},b} \frac{1}{n}\sum_{i=1}^{n} L(h(x_i;\mathbf{w},b),y_i)$$

General recipe

- Fix hypothesis class

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

- Define loss function

$$L(h(x; \mathbf{w}, b), y) = -y \log h(x; \mathbf{w}, b) + (1 - y) \log(1 - h(x; \mathbf{w}, b))$$

- Minimize average loss on the training set

$$\min_{\mathbf{w}, b} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i; \mathbf{w}, b), y_i)$$

- *Why should this work?*
- *How do we do the minimization in practice*

Training = Optimization

- Need to minimize an objective

$$\min_{\mathbf{w},b} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i; \mathbf{w}, b), y_i)$$

- More generally, objective takes the form

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i, \boldsymbol{\theta}) \equiv \min_{\boldsymbol{\theta}} F(\boldsymbol{\theta})$$

Training = optimization

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i, \boldsymbol{\theta}) \quad \equiv \min_{\boldsymbol{\theta}} F(\boldsymbol{\theta})$$

- How do we minimize this?
- Start from an initial estimate
- Iteratively reduce F. How?

Optimization and function gradients

- Suppose current estimate is $\boldsymbol{\theta}^{(t)}$
- Consider changing this to $\boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}$
- How does the objective value change?
- For small $\Delta\boldsymbol{\theta}$, can approximate F using Taylor expansion
  - F is *locally linear*

$$F(\boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}) \approx F(\boldsymbol{\theta}^{(t)}) + \nabla F(\boldsymbol{\theta}^{(t)})^T \Delta\boldsymbol{\theta}$$

$$\Rightarrow F(\boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}) - F(\boldsymbol{\theta}^{(t)}) \approx \nabla F(\boldsymbol{\theta}^{(t)})^T \Delta\boldsymbol{\theta}$$

Optimization and function gradients

$$\Rightarrow F(\boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}) - F(\boldsymbol{\theta}^{(t)}) \approx \nabla F(\boldsymbol{\theta}^{(t)})^T \Delta\boldsymbol{\theta}$$

- We want $F(\boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}) - F(\boldsymbol{\theta}^{(t)})$ to be negative
    - As highly negative as possible
- So we want $\nabla F(\boldsymbol{\theta}^{(t)})^T \Delta\boldsymbol{\theta}$ to be as negative as possible

$$\Delta\boldsymbol{\theta} = -\lambda \nabla F(\boldsymbol{\theta}^{(t)})$$

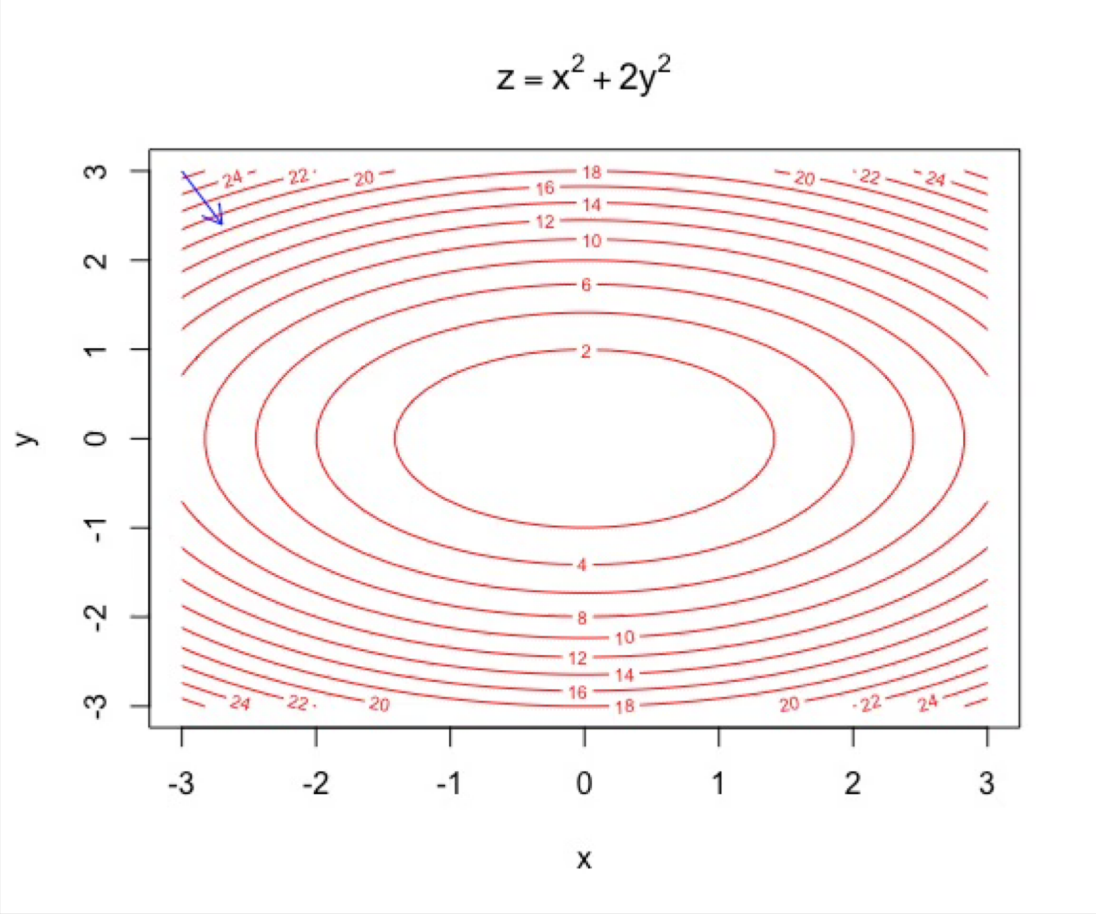
$$\Rightarrow \nabla F(\boldsymbol{\theta}^{(t)})^T \Delta\boldsymbol{\theta} = -\lambda \|\nabla F(\boldsymbol{\theta}^{(t)})\|^2$$

- λ is step size

Optimization using gradient descent

- Randomly initialize $\boldsymbol{\theta}^{(0)}$
- For i = 1 to max_iterations:
  - Compute gradient of F at $\boldsymbol{\theta}^{(t)}$
  - $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \lambda \nabla F(\boldsymbol{\theta}^{(t)})$
    - Function value will decrease by $\lambda||\nabla F\left(\boldsymbol{\theta}^{(t)}\right)||^2$
  - Repeat until $||\nabla F\left(\boldsymbol{\theta}^{(t)}\right)||^2$ drops below a threshold

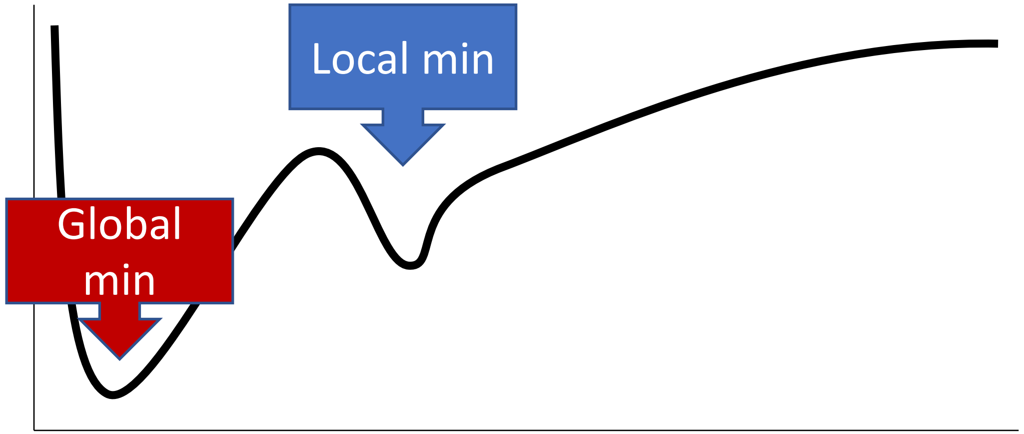
Gradient descent



https://yihui.name/animation/example/grad-desc/

Gradient descent - convergence

- Every step leads to a reduction in the function value
- If function is bounded below, we will eventually stop
- But will we stop at the right “global minimum”?
  - Not necessarily: local optimum!



Gradient descent in machine learning

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} f(x_i, y_i, \boldsymbol{\theta}) \qquad \equiv \min_{\boldsymbol{\theta}} F(\boldsymbol{\theta})$$

$$\nabla F(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f(x_i, y_i, \boldsymbol{\theta})$$

- Computing the gradient requires a *loop over all training examples*
- Very expensive for large datasets

Stochastic gradient descent

$$\nabla F(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f(x_i, y_i, \boldsymbol{\theta})$$

- Gradient is *average* of per-example gradient
- Can get an *estimate* of the average by using a small random sample (called a “minibatch”)

$$\nabla F(\boldsymbol{\theta}) \approx \frac{1}{k} \sum_{j=1}^{k} \nabla f(x_{i_j}, y_{i_j}, \boldsymbol{\theta})$$

- Take step along estimated gradient
- *Stochastic gradient descent!*

General recipe

Logistic Regression!

- Fix hypothesis class

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

- Define loss function

$$L(h(x; \mathbf{w}, b), y) = -y \log h(x; \mathbf{w}, b) + (1 - y) \log(1 - h(x; \mathbf{w}, b))$$

- Minimize average loss on the training set using SGD

$$\min_{\mathbf{w}, b} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i; \mathbf{w}, b), y_i)$$

General recipe

- Fix hypothesis class

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

- Define loss function

$$L(h(x; \mathbf{w}, b), y) = -y \log h(x; \mathbf{w}, b) + (1 - y) \log(1 - h(x; \mathbf{w}, b))$$

- Minimize average loss on the training set using SGD

$$\min_{\mathbf{w}, b} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i; \mathbf{w}, b), y_i)$$

- *Why should this work?*

Why should this work?

- Let us look at the objective more carefully

$$\min_{\mathbf{w},b} \frac{1}{n}\sum_{i=1}^{n} L(h(x_i;\mathbf{w},b),y_i)$$

- We are minimizing average loss on the training set
- Is this what we actually care about?

Risk

- Given:
  - Distribution over (x,y) pairs
  - A hypothesis $\mathcal{D}$ from hypothesis class H
  - Loss function L $h \in H$
- We are interested in Expected Risk (think of this as "Error"):

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y)$$

- Given training set S, and a particular hypothesis h, Empirical Risk (Training error):

$$\hat{R}(S, h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

Risk

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y) \qquad \hat{R}(S,h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

- Left: true quantity of interest, right: estimate
- How good is this estimate?
- If h is *randomly chosen,* actually a pretty good estimate!
  - In statistics-speak, it is an *unbiased estimator* : correct in expectation

$$\mathbb{E}_{S\sim\mathcal{D}^n} \hat{R}(S,h) = R(h)$$

Risk

- Empirical risk unbiased estimate of expected risk
- Want to minimize expected risk
- Idea: Minimize *empirical risk* instead
- This is the **Empirical Risk Minimization Principle**

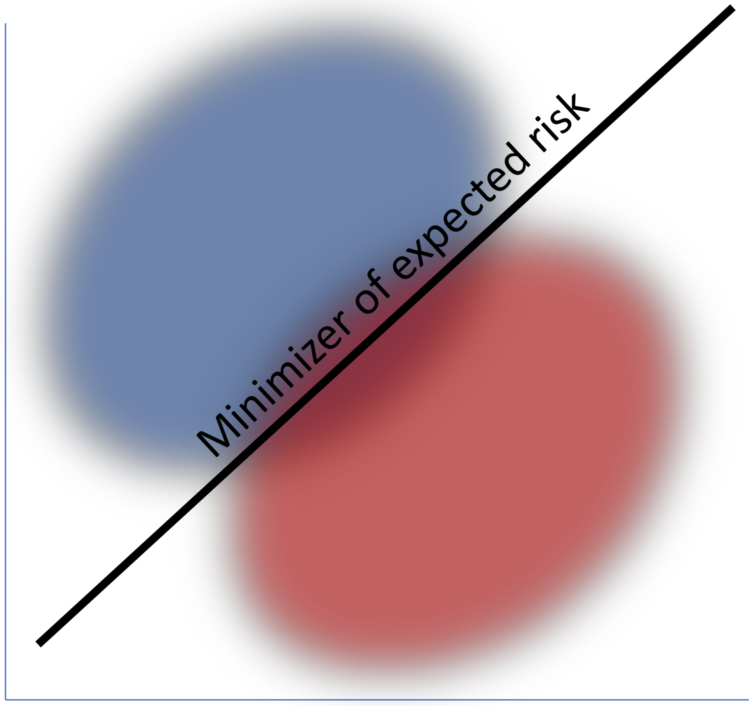
$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y) \qquad \hat{R}(S,h) = \frac{1}{|S|} \sum_{(x,y)\in S} L(h(x), y)$$

$$\boxed{h^* = \arg\min_{h\in H} \hat{R}(S,h)}$$

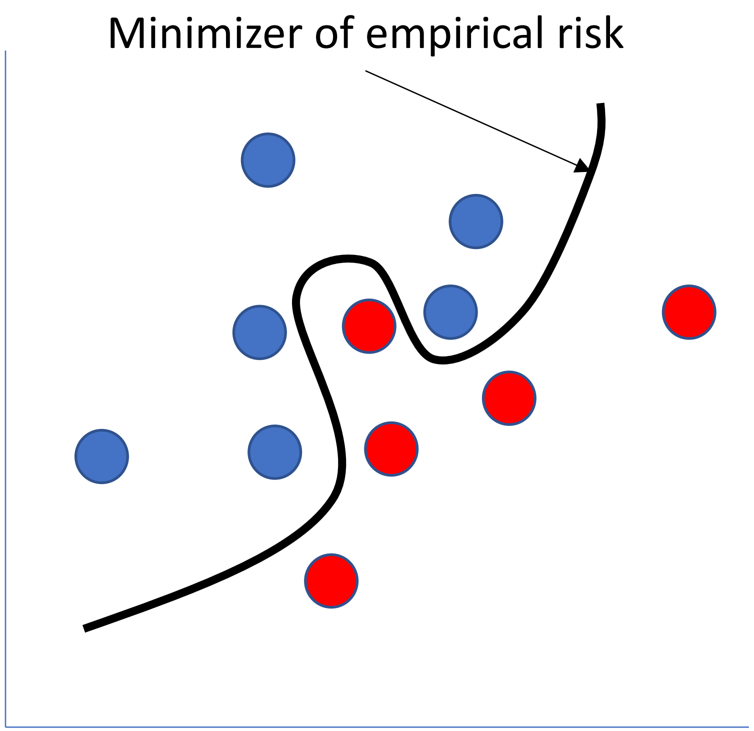
Overfitting

- For *randomly chosen* h, empirical risk (training error) good estimate of expected risk
- But we are *choosing* h by minimizing training error
- Empirical risk of chosen hypothesis *no longer* unbiased estimate:
  - We chose hypothesis based on S
  - Might have chosen h for which S is a special case
- Overfitting:
  - Minimize training error, but generalization error *increases*

Overfitting = fitting the noise



True distribution



Sampled training set

Generalization

$$R(h) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(h(x), y) \qquad \hat{R}(S,h) = \frac{1}{|S|}\sum_{(x,y)\in S} L(h(x), y)$$

$$R(h) = \underbrace{\hat{R}(S,h)}_{\text{Training error}} + \underbrace{(R(h) - \hat{R}(S,h))}_{\text{Generalization error}}$$

Controlling generalization error

- Variance of empirical risk inversely proportional to size of S (central limit theorem)
  - Choose very large S!
- *Larger* the hypothesis class H, *Higher* the chance of hitting bad hypotheses with low training error and high generalization error
  - Choose small H!
- For many models, can *bound* generalization error using some property of parameters
  - “Regularization”

Controlling the size of the hypothesis class

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

- How many parameters (**w,** b) are there to find?
- Depends on dimensionality of ϕ
- Large dimensionality = large number of parameters = more chance of overfitting
- Rule of thumb: size of training set should be at least 10x number of parameters
- Often training sets are much smaller

Regularization

- Old objective

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} L(h(x_i; \mathbf{w}, b), y_i)$$

- New objective

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} L(h(x_i; \mathbf{w}, b), y_i) + \lambda \|\mathbf{w}\|^2$$

- Why does this help?

Regularization

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} L(h(x_i; \mathbf{w}, b), y_i) + \lambda \|\mathbf{w}\|^2$$

- Ensures classifier does not weigh any one feature too highly
- Makes sure classifier scores *vary slowly* when image changes

$$|\mathbf{w}^T \phi(x_1) - \mathbf{w}^T \phi(x_2)| \leq \|\mathbf{w}\| \|\phi(x_1) - \phi(x_2)\|$$

Controlling generalization error

- How do we know we are overfitting?
  - Use a *held-out* “validation set”
  - To be an unbiased sample, must be completely *unseen*
- Choose hypothesis class, regularization etc that lowers validation error
- Note: to get final estimate of expected risk, need *another* held-out set: the “test set”

Putting it all together

- Want model with least expected risk = expected loss
- But expected risk hard to evaluate
- Empirical Risk Minimization: minimize empirical risk in training set
- Might end up picking special case: overfitting
- Avoid overfitting by:
  - Constructing large training sets
  - Reducing size of model class
  - Regularization

Putting it all together

- Collect training set and validation set
- Pick hypothesis class
- Pick loss function
- Minimize empirical risk (+ regularization)
- Measure performance on held-out validation set
- Profit!

Loss functions and hypothesis classes

Loss function	Problem	Range of h	$\mathcal{Y}$	Formula
Log loss	Binary Classification	$\mathbb{R}$	$\{0, 1\}$	$\log(1 + e^{-yh(x)})$
Negative log likelihood	Multiclass classification	$[0, 1]^k$	$\{1, \ldots, k\}$	$-\log h_y(x)$
Hinge loss	Binary Classification	$\mathbb{R}$	$\{0, 1\}$	$\max(0, 1 - yh(x))$
MSE	Regression	$\mathbb{R}$	$\mathbb{R}$	$(y - h(x))^2$

Back to images

$$h(x; \mathbf{w}, b) = \sigma(\mathbf{w}^T \phi(x) + b)$$

- What should ϕ be?
- Simplest solution: string 2D image intensity values into vector