Machine Learning Overview

Yale

CPSC/AMTH/CBB 663

CPSC 452

xkcd.com/1838

Announcements

- Tomorrow at 3pm: Kincaid tutorial on Getting Set Up
- Sunday 9-11am: Creating a neural network in pytorch
  - See blog: https://towardsdatascience.com/tabnet-deep-neural-network-for-structured-tabular-data-39eb4b27a9e4
- Office hours begin next week!
- Homework 1 will be assigned TONIGHT, due Feb 15
- If you have questions to ask me, please ask via discord
  - ➔ JOIN DISCORD!!
- Class has a partition on the HPC Server Farnam (see canvas announcement)

what does it mean for a machine or a program learn?

① minimize error on unseen data
② tune params to a model to better fit the data
③ some set of params are updated to improve results of a target function
④ decode the features of a dataset
⑤ find latent pattern in data without explicit instruction
⑤ minimize cost function doesn't mean you are learning
⑥ teach itself to perform better
⑦ find pattern or make prediction for data

- Machine learning algorithm = algorithm that can learn from data ≠ meta-learning
- What do we mean by "learn from data"?
- "A computer program is said to learn from : canonical def ML
  - <u>experience</u> E
  - with respect to some class of <u>tasks</u> T
  - <u>performance measure</u> P
- **IF** its performance at tasks in T, as measured by P, improves with experience E." – Mitchell (1997)
- What are E, T, and P?
  - It depends on the problem!

some abstract problems are hard to find P

data: text, image, numerical, tabular

task: human can do

P: same data + different P → different solutions

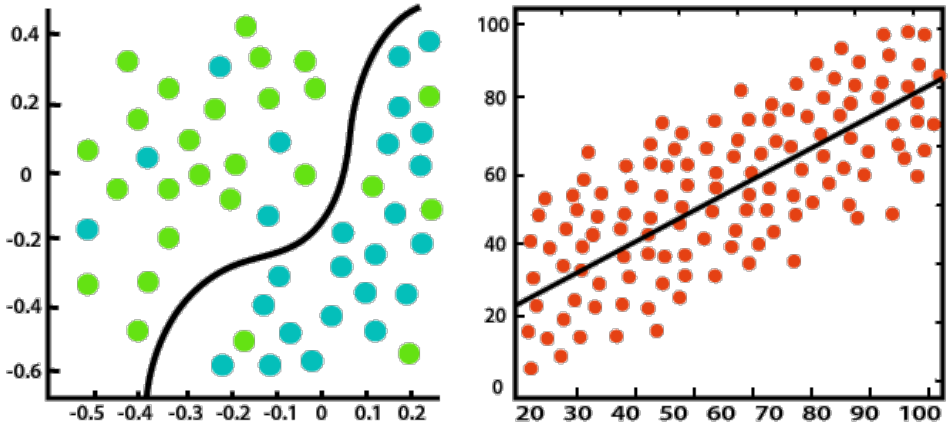
- Machine learning helps us to solve problems that are too difficult for fixed programs
- Learning is the process of attaining the ability to perform the task
- Tasks are typically described in terms of how to process an ***example***
  - An example is a collection of measured ***features***
  - Often represented as a vector $x \in \mathbb{R}^d$ of features
  - E.g. an example includes a single image with the features comprised of the pixels in the image

often in high dim

- Classification

draw a boundary in a feature space

  - Output: a function $f: \mathbb{R}^d \rightarrow \{1, \ldots, k\}$
- Regression
  - Output: a function $f: \mathbb{R}^d \rightarrow \mathbb{R}$



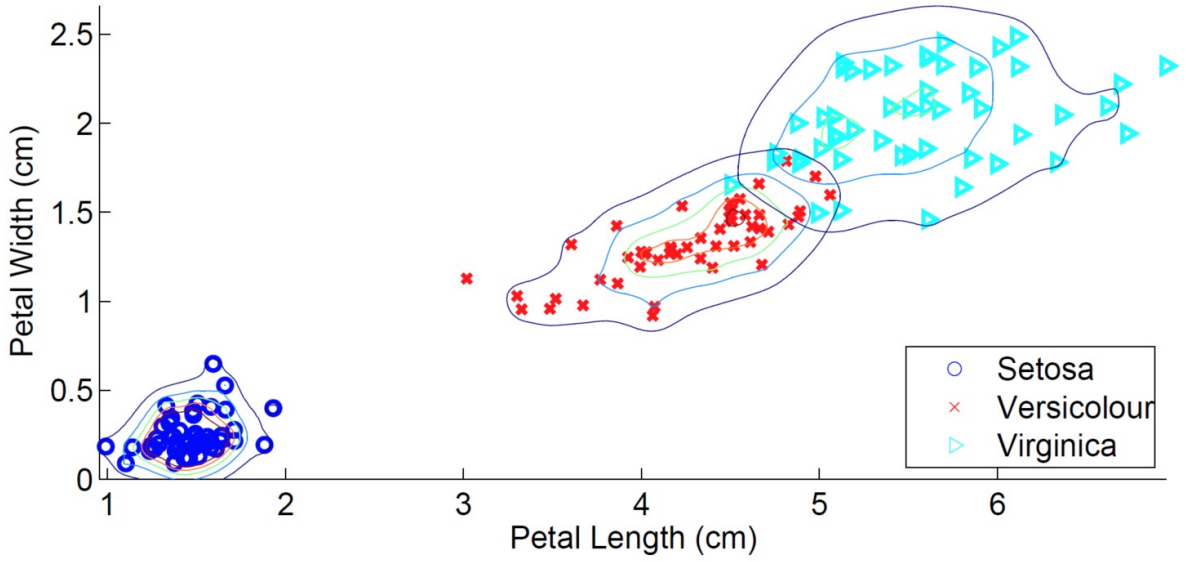
Classification Regression

Linear regression example

- Need a quantitative measure to evaluate a machine learning algorithm
- Provides a measure to guide the selection of an algorithm
- Examples:
  - **A*ccuracy*** (or equivalently, the ***error rate***) in a classification problem
  - Prediction error
- Performance is typically ***estimated***/evaluated based on a *test set* of data (different from training data)
- Choosing a performance measure isn't always straightforward

- Most algorithms **experience** an entire ***dataset*** consisting of many examples or ***data points***
- Example: Iris data set (Fisher, 1936)
  - Four features: sepal length, sepal width, petal length, petal width
  - 3 species/classes, 50 data points per class

clear decision boundary



- Goal is to build a program that learns
  - from experience E
  - to do well at task T
  - as measured by performance measure P
- Correctly predicting the right classes
  - How would this be a performance measure?
  - Is it differentiable?

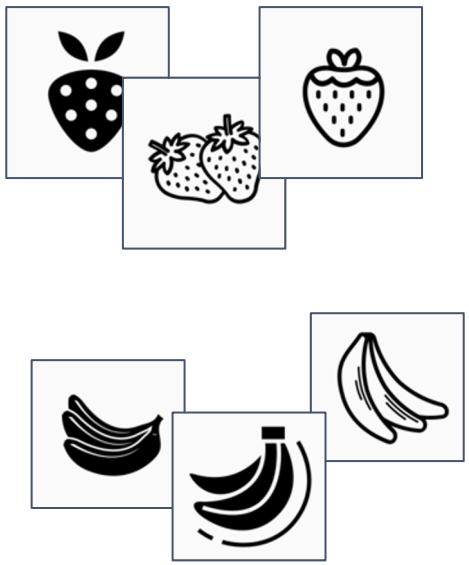
we can't use accuracy because it jumps

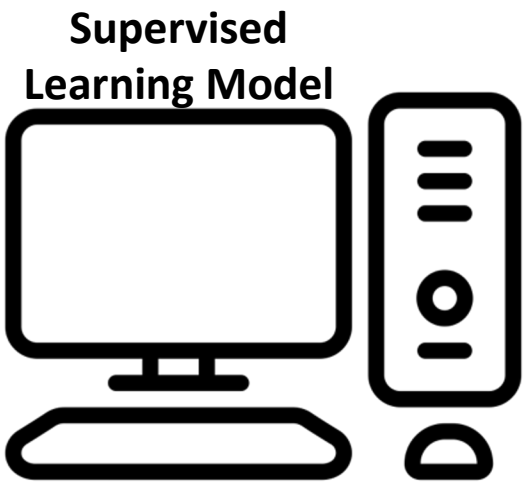
often cross-entropy loss

Supervised Learning

Supervised learning

- Have a bunch of labelled data, want to label new data

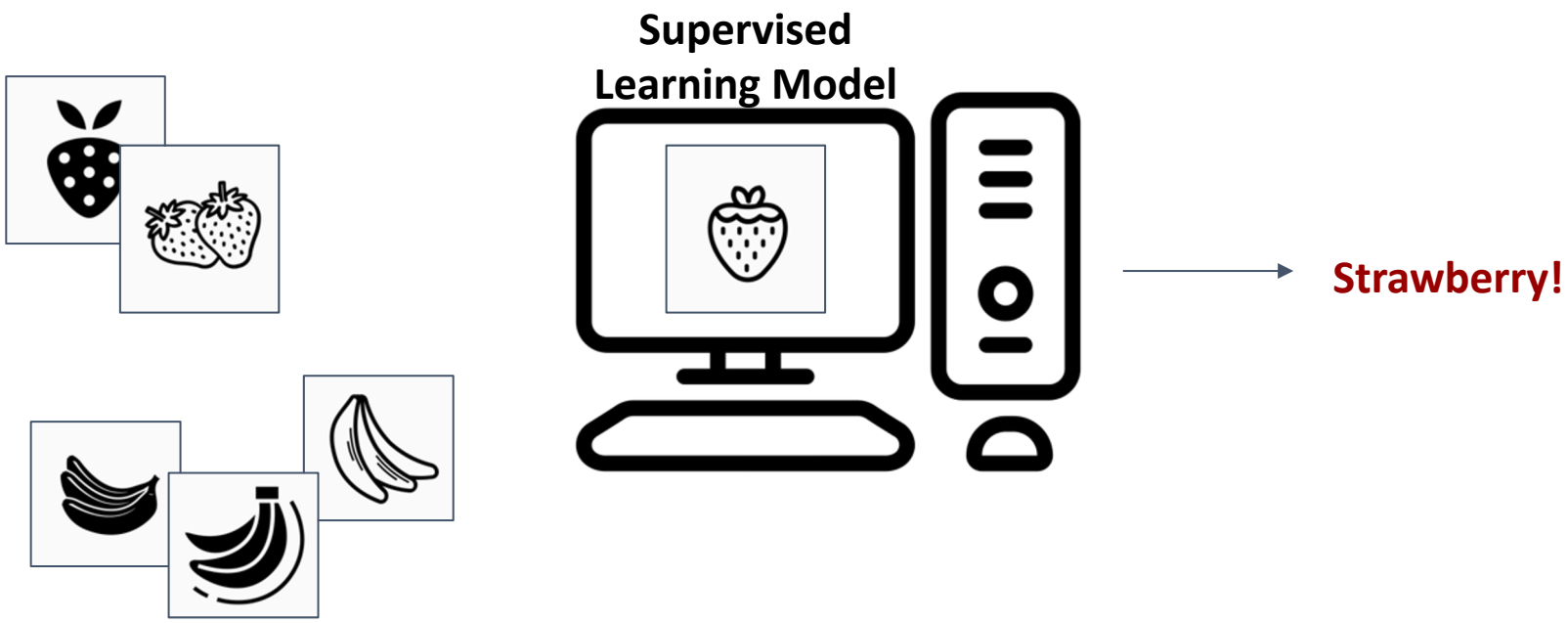




Noun project

Supervised learning

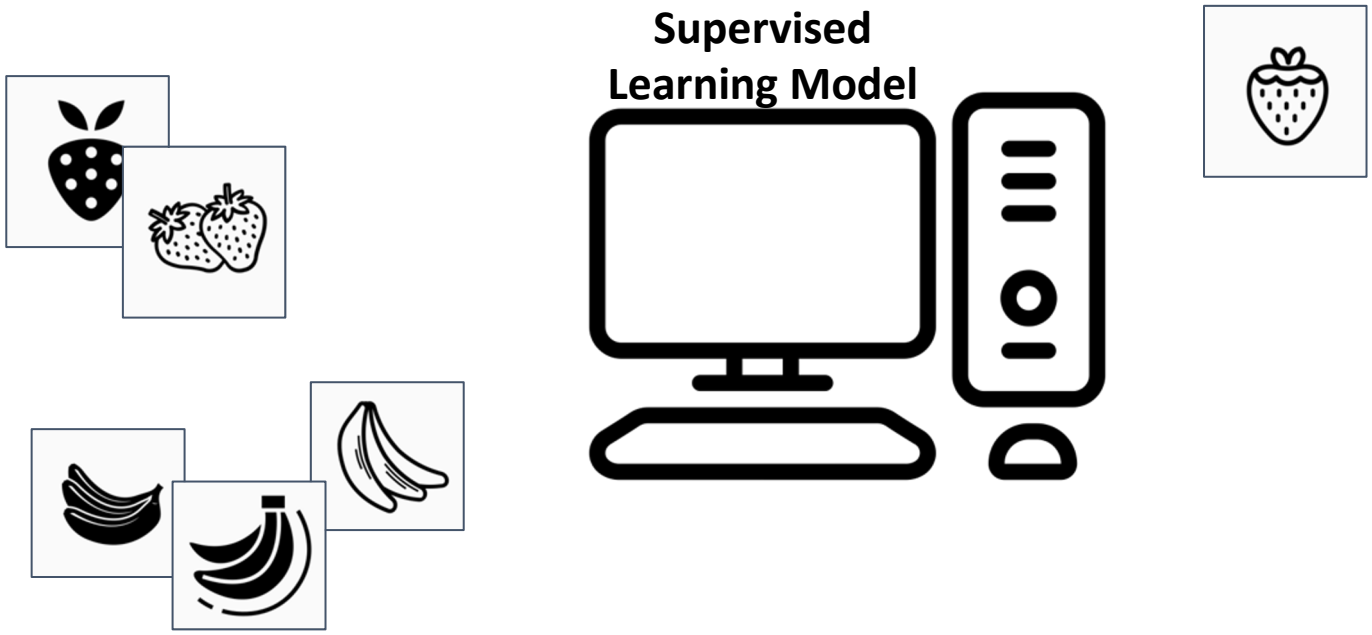
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

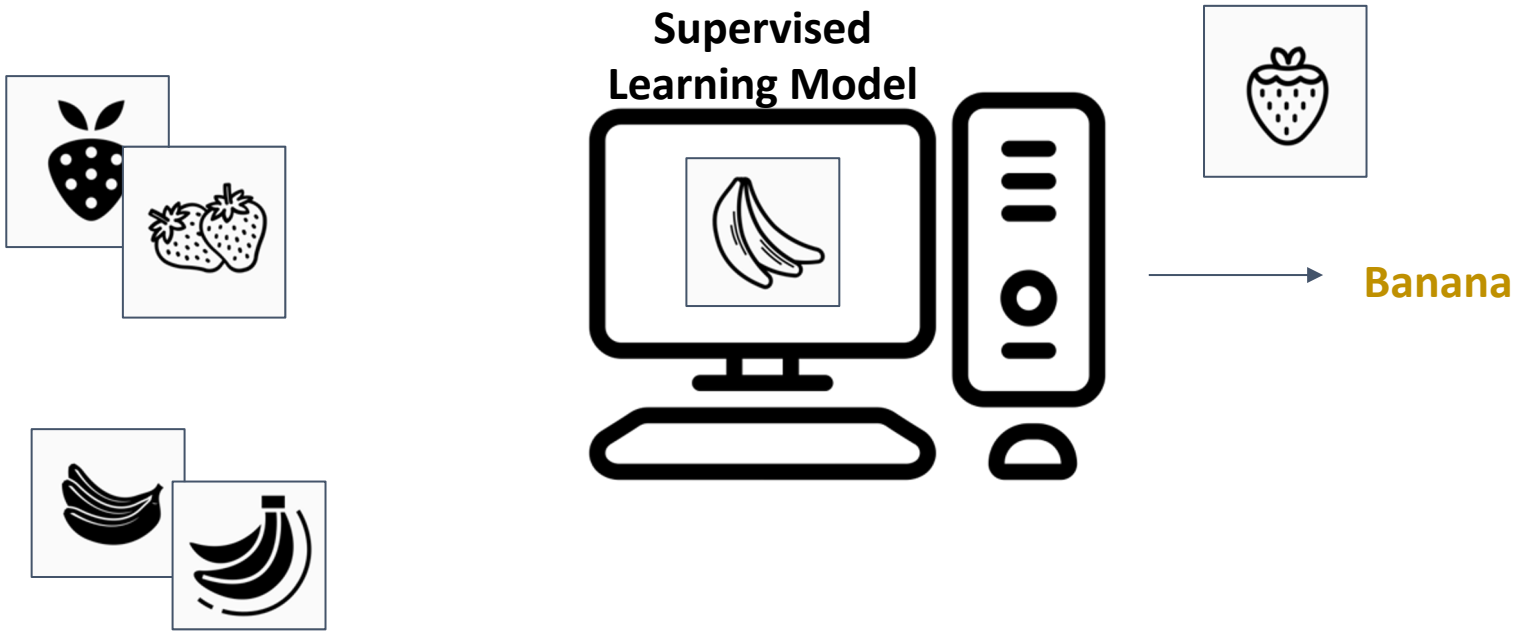
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

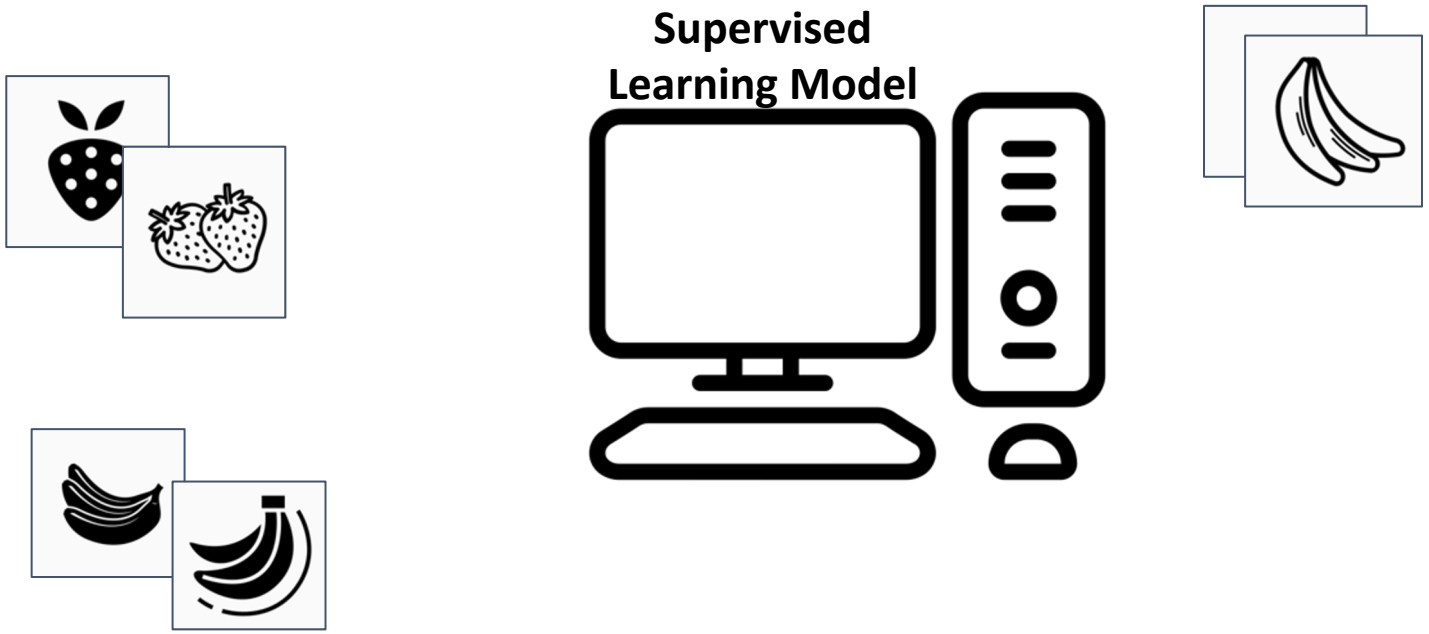
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

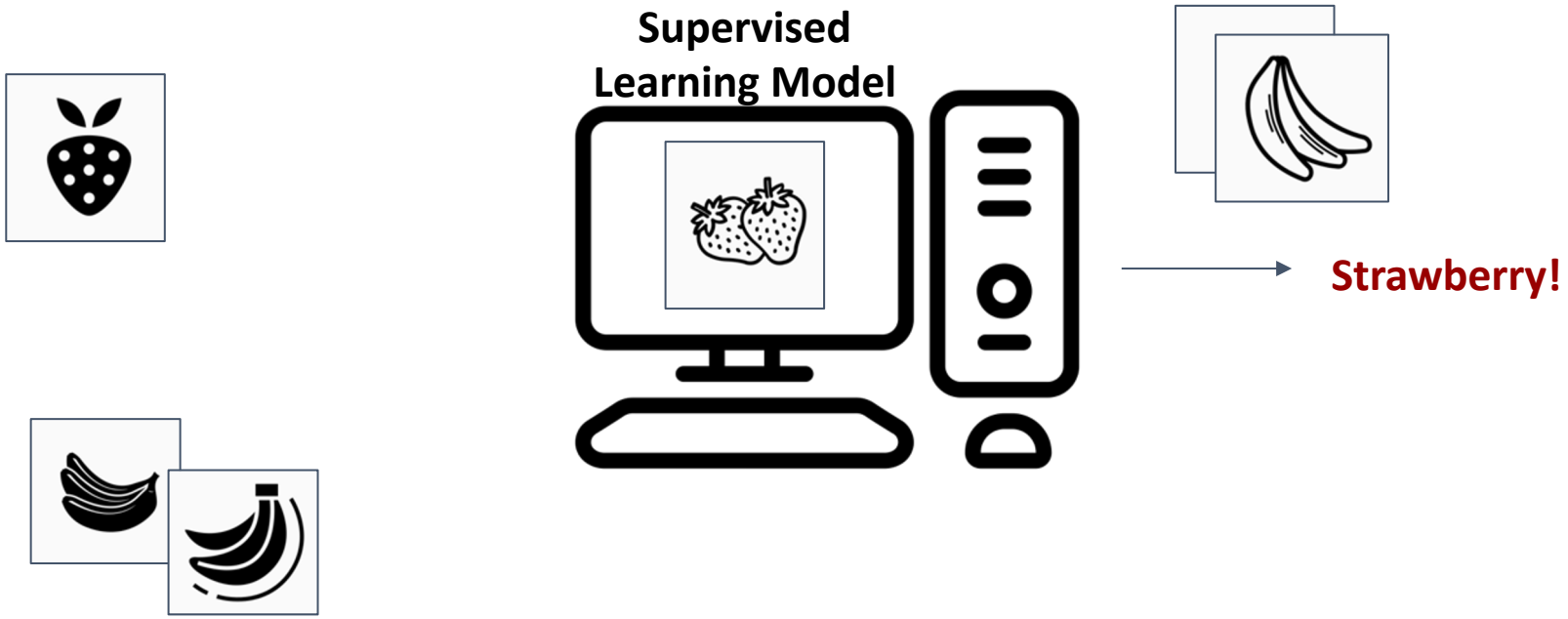
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

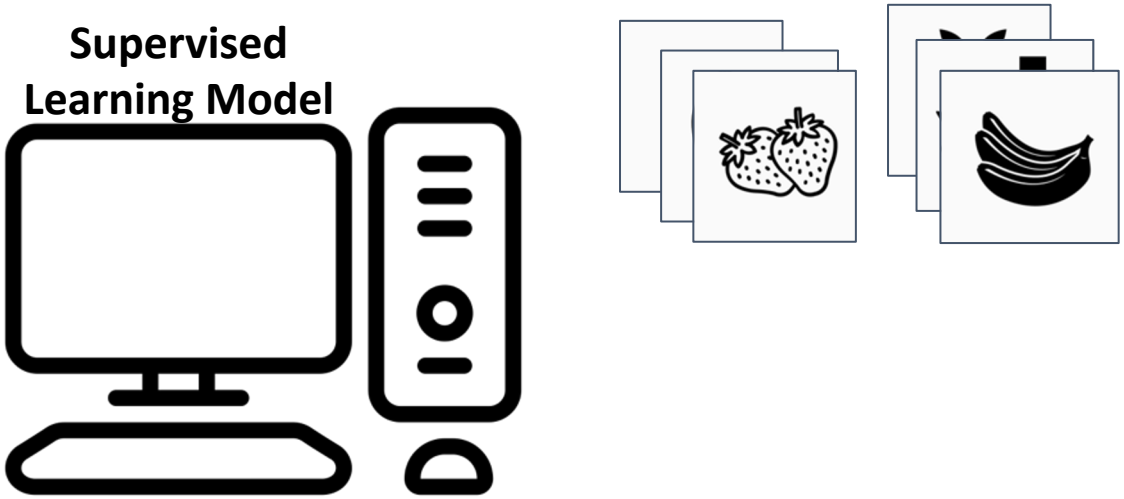
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

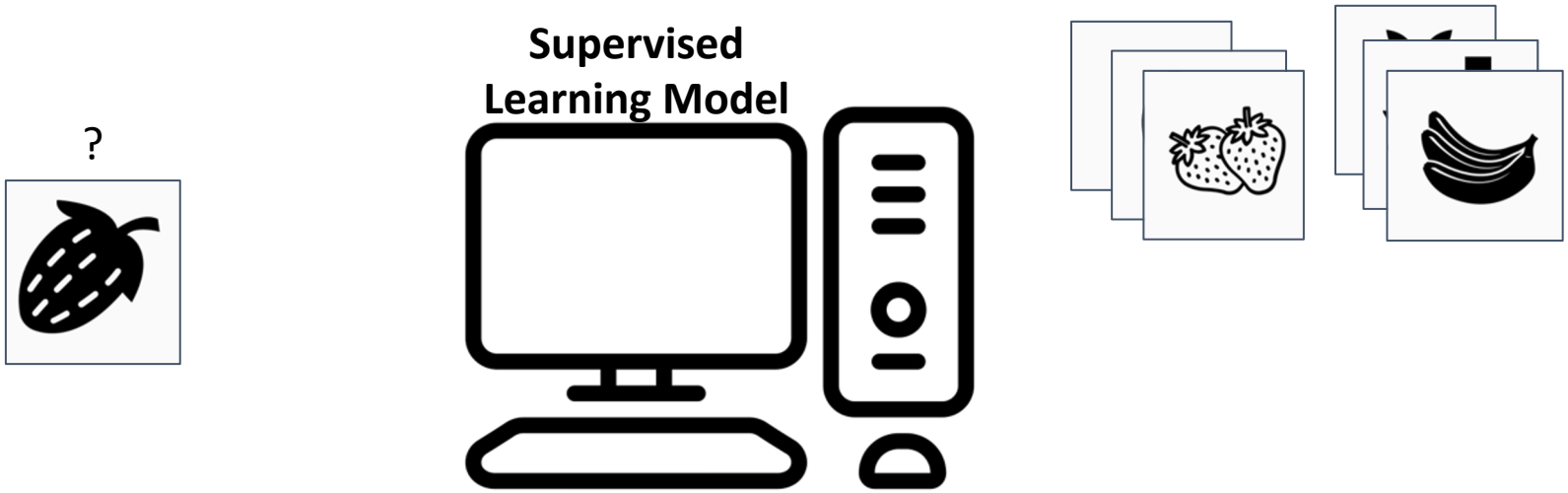
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

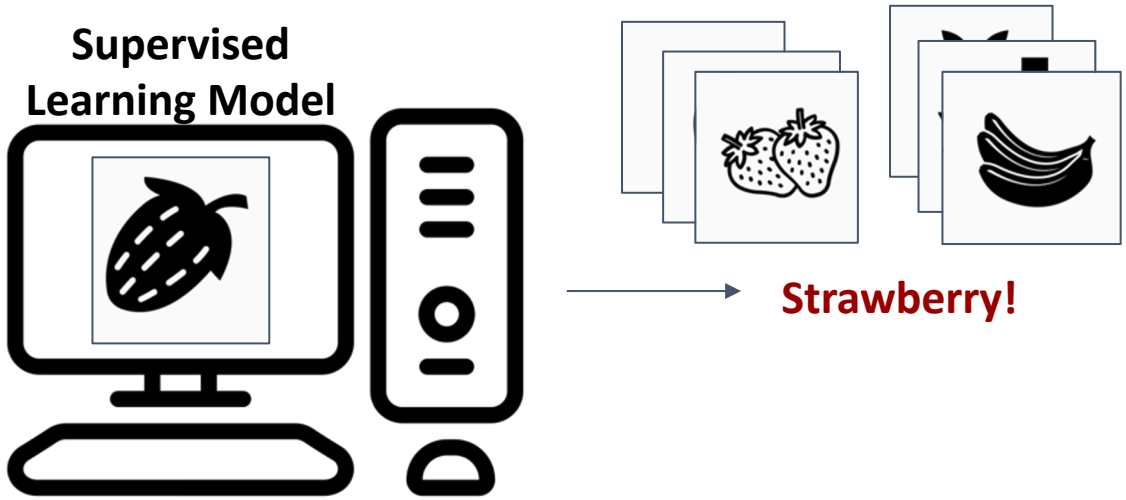
- Have a bunch of labelled data, want to label new data



Noun project

Supervised learning

- Have a bunch of labelled data, want to label new data



Noun project

- Dataset contains features and a ***label*** or target
  - E.g. classify iris plants into the three different species
- Mathematically, supervised learning observes examples of a random vector $\boldsymbol{x}$ and an associated label $\boldsymbol{y}$
  - Goal is to learn to predict $\boldsymbol{y}$ from $\boldsymbol{x}$

"supervised"

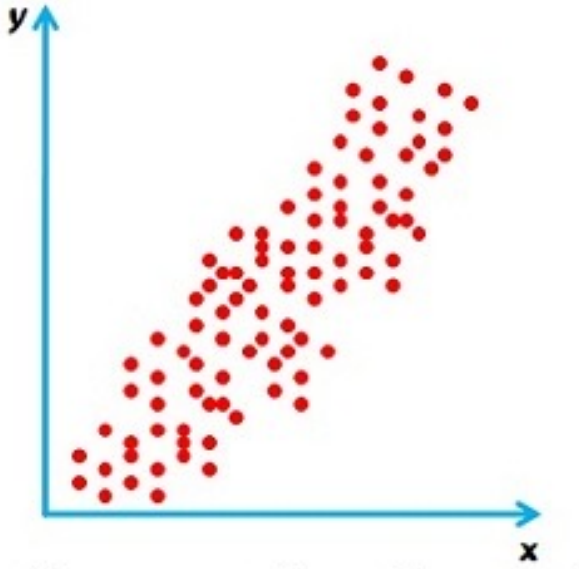
- Term comes from view of the label $\boldsymbol{y}$ coming from an instructor or teacher

Supervised learning

- Input data space $\mathcal{X}$
- Output (label) space $\mathcal{Y}$
- Unknown function $f: \mathcal{X} \rightarrow \mathcal{Y}$ (mapping)
- Dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)\}$ with $\boldsymbol{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$
- Finite $\mathcal{Y} \Rightarrow$ classification
- Continuous $\mathcal{Y} \Rightarrow$ regression

Regression

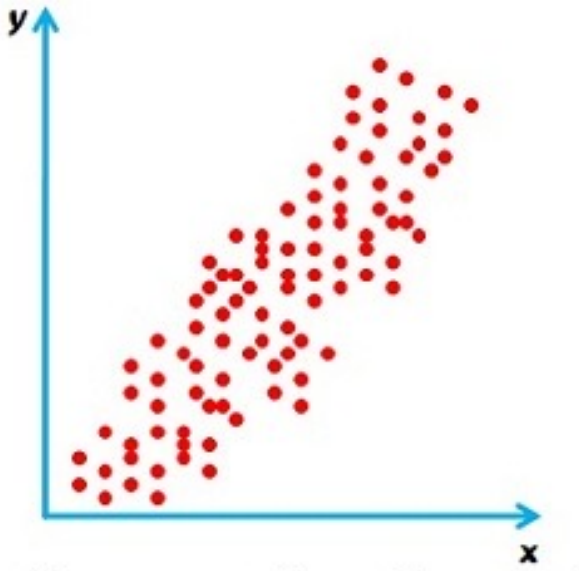
- We have a set of n observations $(\boldsymbol{x}_i, y_i)$ with $y_i \in \mathbb{R}$
- Goal is to learn a function that predicts label y from sample $\boldsymbol{x}$



Surbhi S, 2016

Regression

1. Choose a ***model class*** of functions (model selection)
   mode class of deep NN = last slide of 1st lecture
2. Choose a performance measure to guide the selection of a function from the model class



Surbhi S, 2016

- We'll begin with a simple model class: linear functions

Linear Regression

- We want to fit a linear function to dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)\}$
- $\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^T \boldsymbol{x}$
  - $\boldsymbol{w} \in \mathbb{R}^d$ is a vector of parameters
  - Line if $d = 1$
  - Plane if $d = 2$
  - Hyperplane in general d
- How do we determine $\boldsymbol{w}$?
- Based on the performance measure P

Loss functions

- Labels are in $\mathcal{Y}$
  - Discrete (classification)
  - Continuous (regression)
- ***Loss function*** $L: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$
- L maps decisions/predictions to a cost.
  - $L(\hat{y}, y)$ is the penalty for predicting $\hat{y}$ when the true label is y
- Standard choice for regression is ***mean squared error*** (MSE)
  - $L(\hat{y}, y) = (\hat{y} - y)^2$
- Absolute loss: $L(\hat{y}, y) = |\hat{y} - y|$

Empirical loss 实验 loss

- Parametric function $f(\boldsymbol{x}; \boldsymbol{w})$
  - E.g. $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$ for regression
- The ***empirical loss*** on a dataset $\mathcal{D}$ is

$$L(\boldsymbol{w}; \mathcal{D}) = \frac{1}{n} \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i)$$

- Least squares minimizes the empirical MSE
- We care about predicting labels for *new* samples
- Does empirical loss minimization help with that?

case of not help : new samples have different distribution with training samples.

Empirical vs. Expected loss

- Assumption: sample and label pairs $(\boldsymbol{x}, y)$ are drawn from the unknown joint distribution $\mathrm{p}(\boldsymbol{x}, y)$
- Data are sampled i.i.d.
- Empirical loss measured on training data

$$L(\boldsymbol{w}; \mathcal{D}) = \frac{1}{n} \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i)$$

- Goal: minimize the ***expected loss*** (aka ***risk***)

$$R(\boldsymbol{w}) = \mathbb{E}[L(f(\boldsymbol{x}; \boldsymbol{w}), y)]$$

Empirical risk (loss) minimization

- Empirical loss

$$L(\boldsymbol{w}; \mathcal{D}) = \frac{1}{n} \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i)$$

- Risk

$$R(\boldsymbol{w}) = \mathbb{E}[L(f(\boldsymbol{x}; \boldsymbol{w}), y)]$$

- If training set is representative of $p(\boldsymbol{x}, y)$, then empirical loss is a suitable proxy for the risk
- Essentially, we estimate $p(\boldsymbol{x}, y)$ and $R(\boldsymbol{w})$ from the empirical distribution of the data

generalizing out of sample is a problem in NN

eg. can't recognize a cat with noise as cat

1. Choose a model class $\mathcal{H}$ of functions $f: \mathcal{X} \rightarrow \mathcal{Y}$
   - Regression example: linear functions parameterized by $\boldsymbol{w}$: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$
2. Select the function in $\mathcal{H}$ that minimizes the empirical risk
   - Linear regression example: minimize empirical squared loss:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2$$

   - How do we find $\boldsymbol{w}^*$?
   - Use calculus

Linear Regression Optimization

- Add an offset w_0: $f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} + \mathrm{w}_0$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} (\boldsymbol{w}^T\boldsymbol{x}_i + w_0 - y_i)^2$$

$$= \arg\min_{\boldsymbol{w}} L(\boldsymbol{w};\mathcal{D})$$

- Set $\frac{\partial L(\boldsymbol{w};\mathcal{D})}{\partial w_i} = 0$ for each i
- Results in $d+1$ equations and $d+1$ unknowns

$x \in R^d$

Linear Regression

- Switching to vector/matrix notation
  - X = data matrix with an additional column of 1's (for the offset)
  - $\boldsymbol{y}$ = vector of labels
  - Prediction: $\hat{\boldsymbol{y}} = X\boldsymbol{w}$
- Resulting solution is

$$\boldsymbol{w}^* = (X^T X)^{-1} X^T \boldsymbol{y}$$

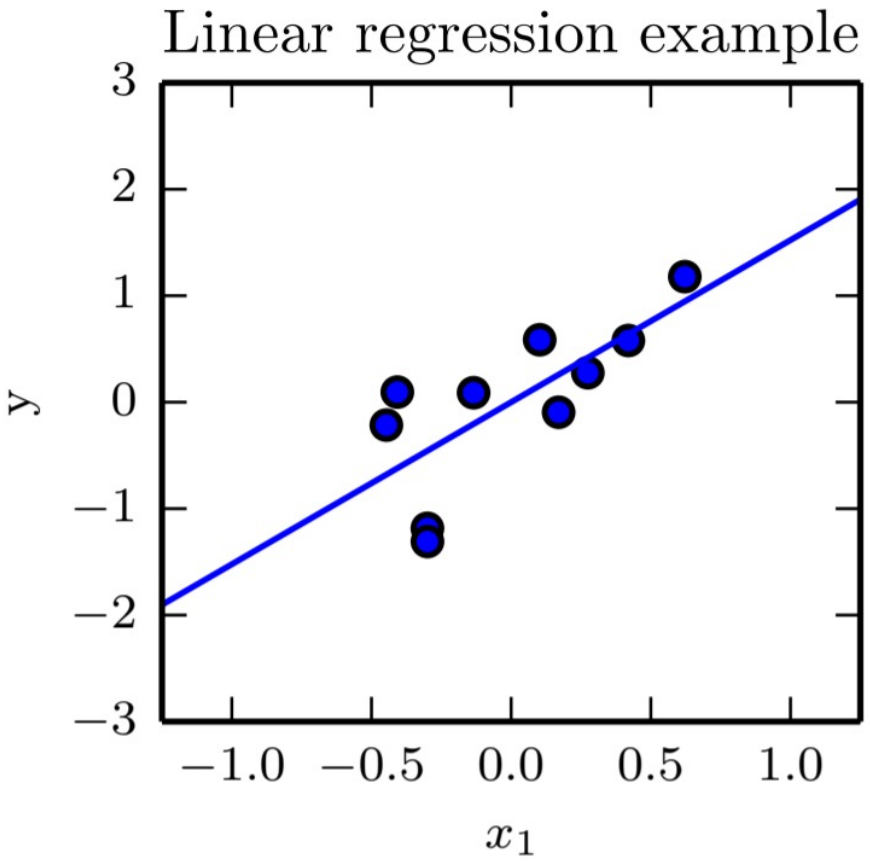
- Closed form solution!
- $(X^T X)^{-1} X^T$ is the Moore-Penrose pseudoinverse of X
- No closed form solution is available in most machine learning

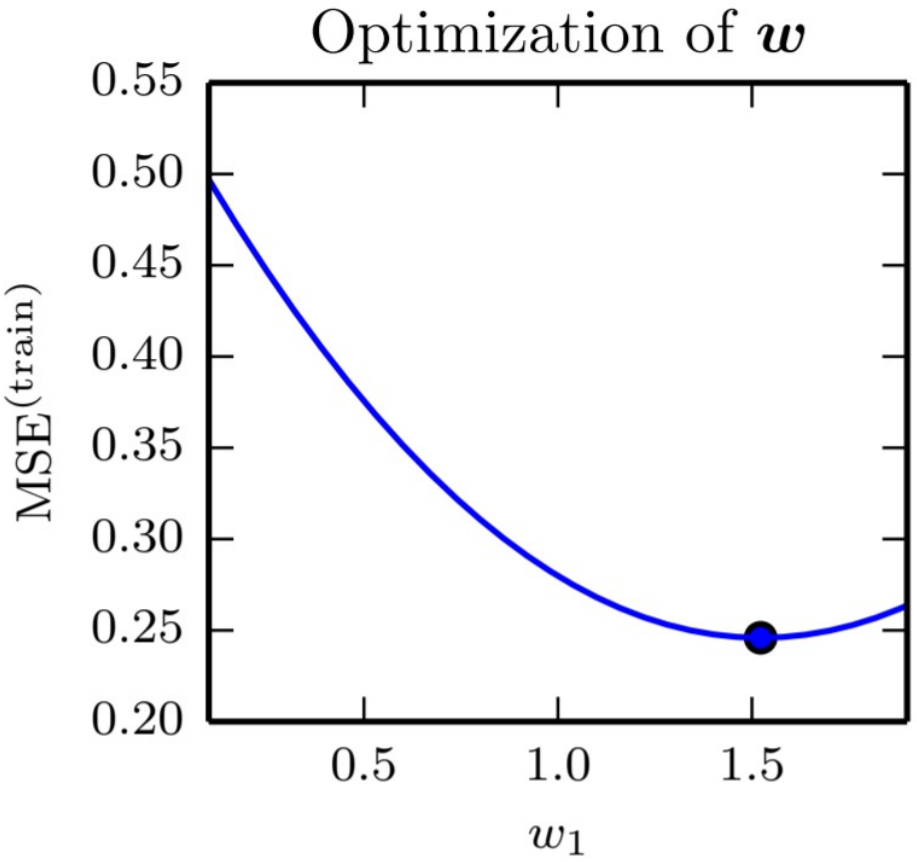
trade-off closed form solution vs function expressiveness

goal: find intercept and slope

theory

optimization is in heart of ML





Goodfellow et al., 2016

generalization

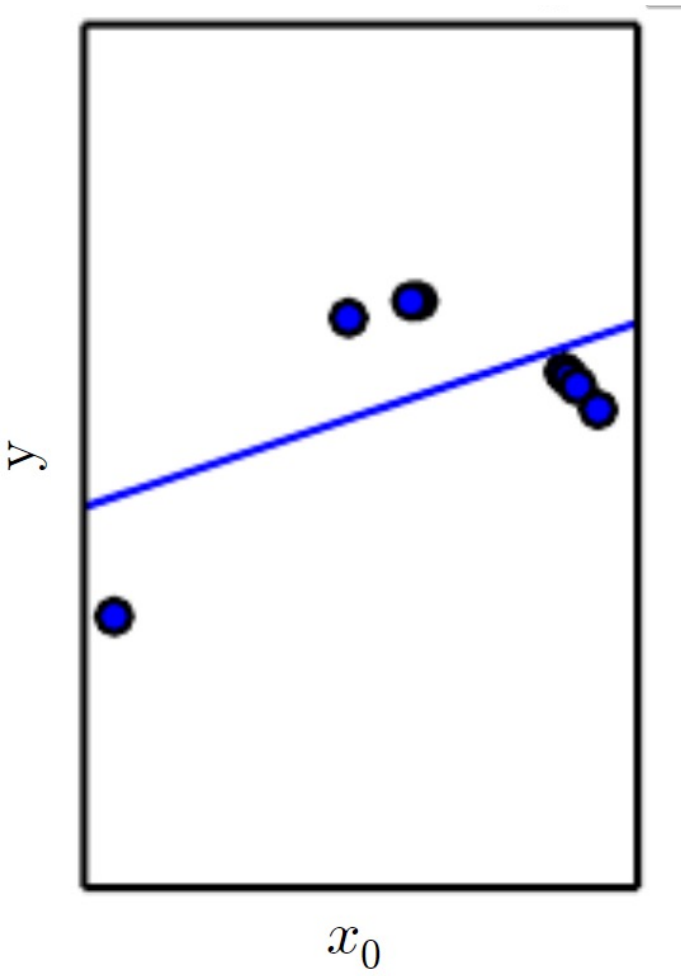
- Challenge of machine learning: algorithm must perform well on previously unseen inputs
  - Not just training inputs
- In other words, we want the algorithm to ***generalize*** well to unseen data
  - ***generalization error*** or the ***test error*** *measures error on unseen data*
- We also need to estimate the generalization error
  - Done typically by dividing the total amount of data into two sets: ***training data*** and ***test data***
  - Model is trained on the training data and then tested on the test data
- Expected test error $\geq$ expected training error

How much ?

Underfitting

Not choose right model class

- Many processes are not well modeled by linear functions
  - Biological systems
  - Stock market prices
  - Periodic signals
- In these cases, linear functions can ***underfit*** the data
  - Training error is very high
  - Test error will also be high



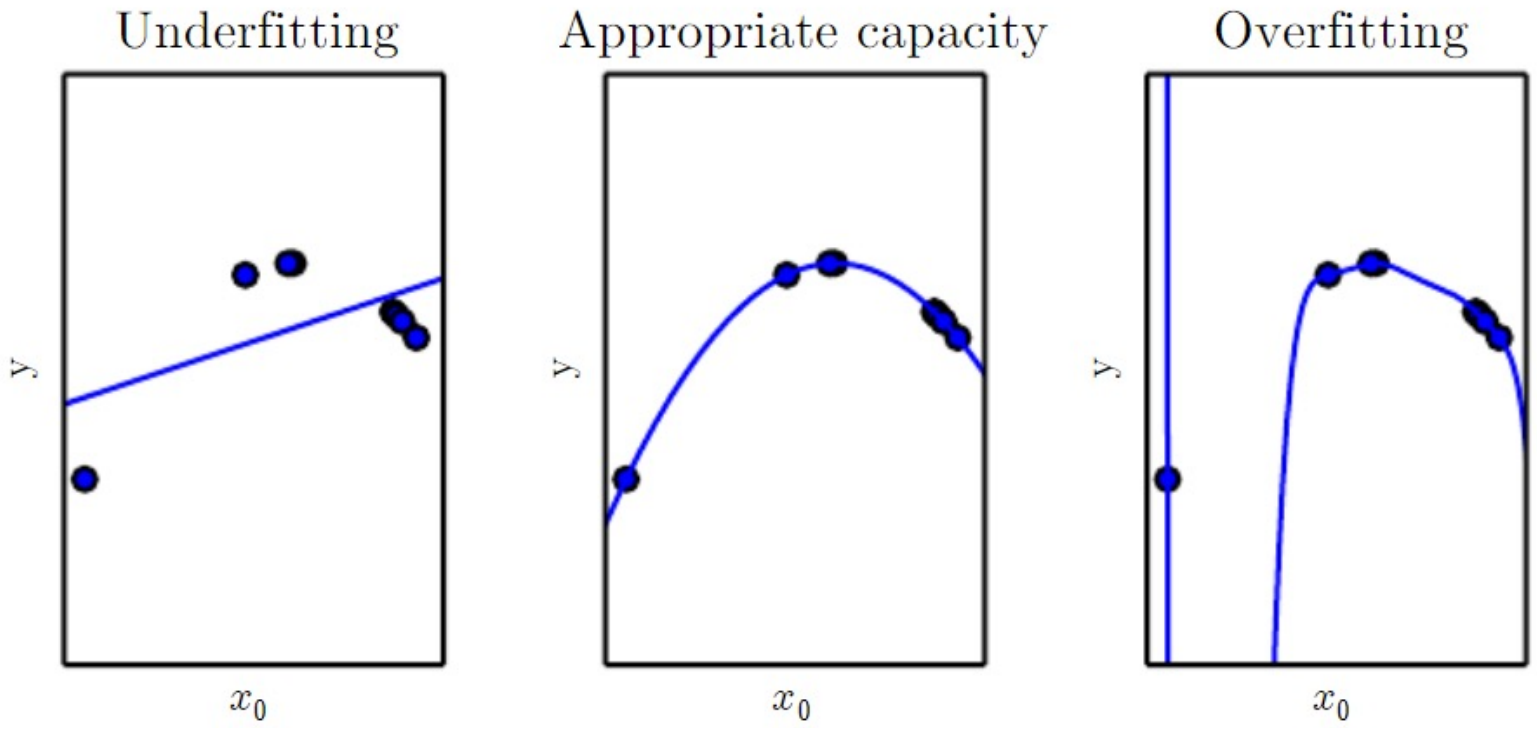
Model Complexity/Capacity

- Traditional Solution: increase ***complexity*** of the model
  - Model complexity is the # of independent parameters in the function model (i.e. "degrees of freedom")
- We can increase the complexity of our linear function by adding polynomial terms
  - Example: $d = 1$

$$f(x; \boldsymbol{w}) = \sum_{j=0}^{m} w_j x^j$$

  - Still linear in $\boldsymbol{w}$ ⇒ we can use linear regression!
- Neural network solution may be to prefer certain types of solutions via regularizations

Nonlinear Regression

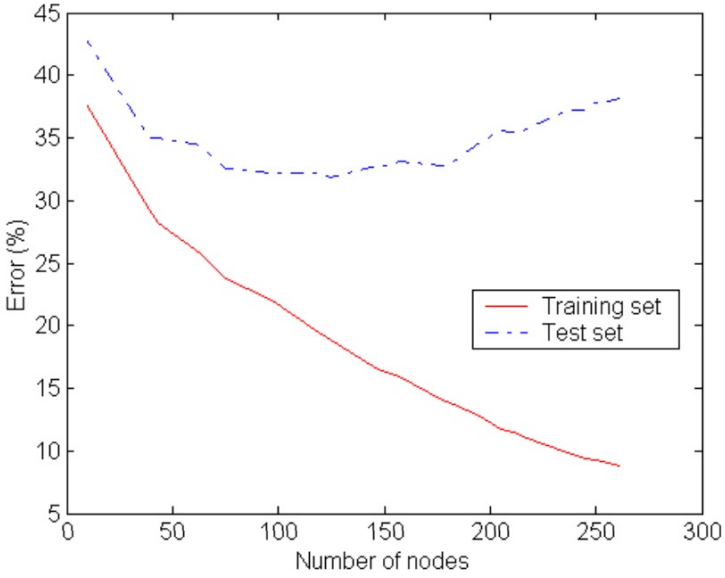


Goodfellow et al., 2016

Overfitting

- Overfitting occurs when the gap between training error and test error is too large
  - The model is capturing unique characteristics of the training set
  - The model does not generalize well to new samples
  - Occurs when the complexity is too large
- Avoiding overfitting
  - Generally requires the use of another test set for evaluation
  - Traditionally: decrease complexity

another solution: regularization



Inductive bias for simplicity

0 coefficients

a preference for more

- Include a penalty on model complexity directly in the cost function:

one kind of regularization

$$\sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \#\text{parameters}$$

- The addition of the penalty is known as ***regularization***
- Regularization: "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." (Goodfellow et al., 2016)
  - Lots of different penalties can be included in regularization
  - Regularization is useful/important in deep learning (and machine learning in general)
- Bias does not have to be directly complexity

regularization express a preference for simplicity in solution, add inductive bias

simplicity { fewer params
smaller range of param values

Regularization

- Ridge regression: penalize with L2 norm

$$\boldsymbol{w}^* = \arg\min \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \lambda \sum_{j=1}^{m} w_j^2$$

prefer smaller coefficient

  - Closed form solution exists $\boldsymbol{w}^* = (\lambda I + X^T X)^{-1} X^T \boldsymbol{y}$ for linear regression

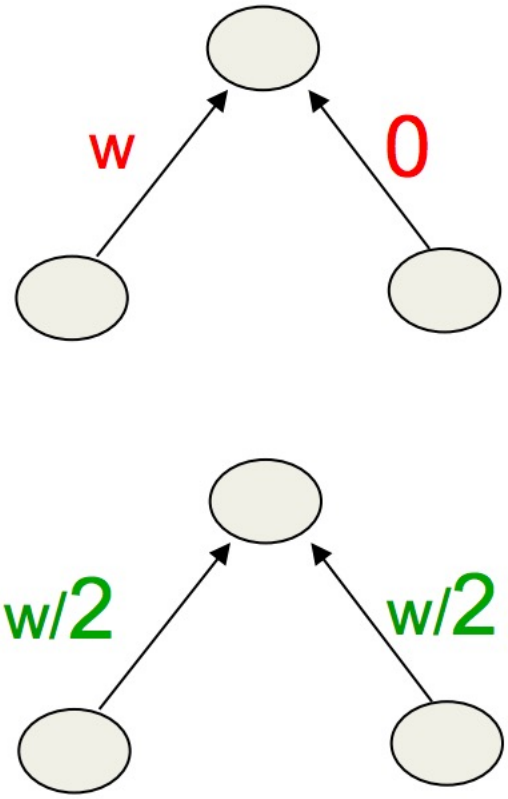
- LASSO regression: penalize with L1 norm

$$\boldsymbol{w}^* = \arg\min \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \lambda \sum_{j=1}^{m} |w_j|$$

  - No closed form solution but still convex (optimal solution can be found) by gradient descent

prefer 0 coefficient

L1 vs L2



- Prefers to share smaller weights
- Makes model smoother
- More Convex

L1 vs L2

L2 regularization	L1 regularization
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases
Non-sparse outputs	Sparse outputs
No feature selection	Built-in feature selection

smaller coefficient made function less fluctuate and smoother

Other regularization

- increase size of training data
- early stopping ✡ good for GAN
- drop out

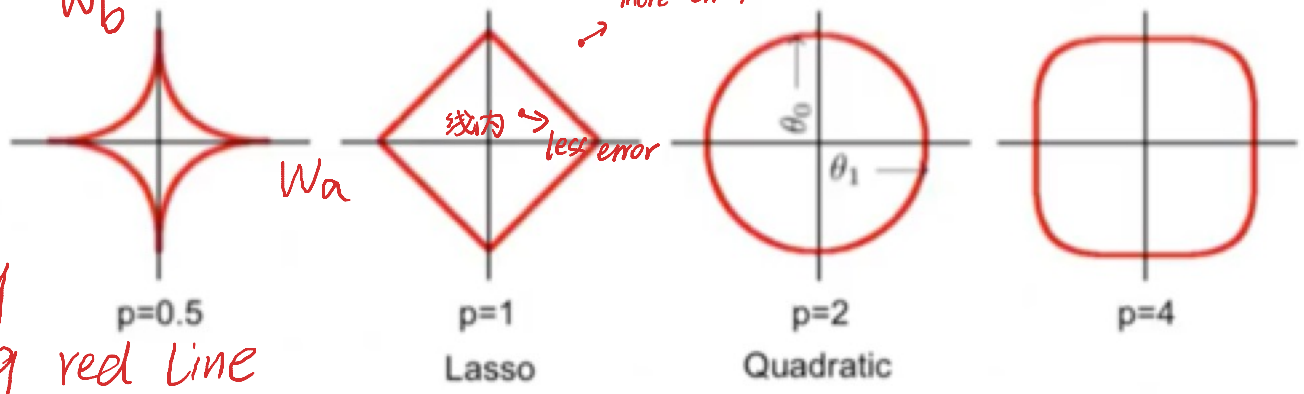
Lp Regularization

$$L_p = (\sum_i w_i^p)^{1/p}$$

assume 2 params a, b

Isosurfaces

等值面



same penalty value along red line

only care about which param has max value

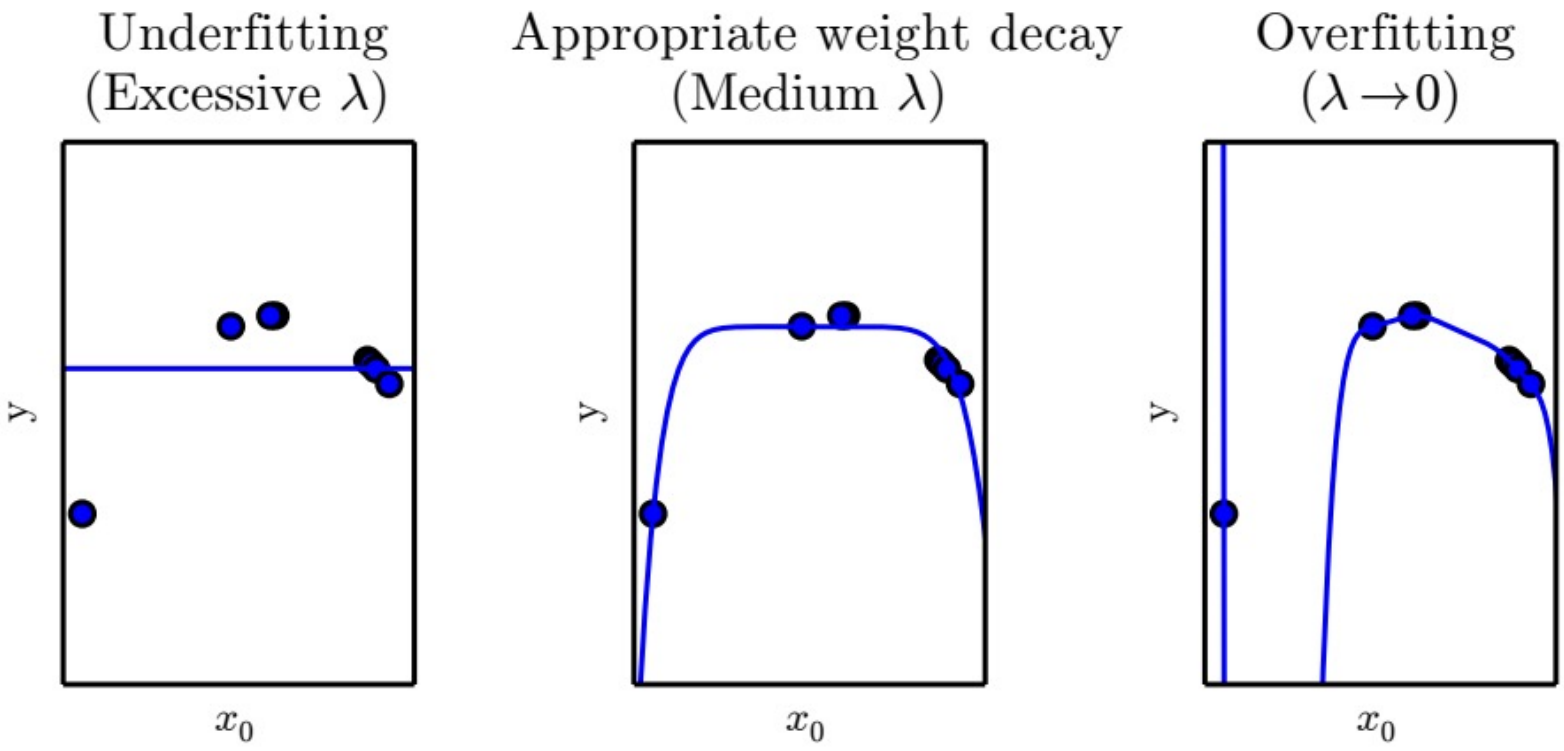
$p \to \infty$ $L_p = \max(w_i)$ $(i = 1, \dots, d)$

goal: # param ↓

As p→0 you get a counting penalty

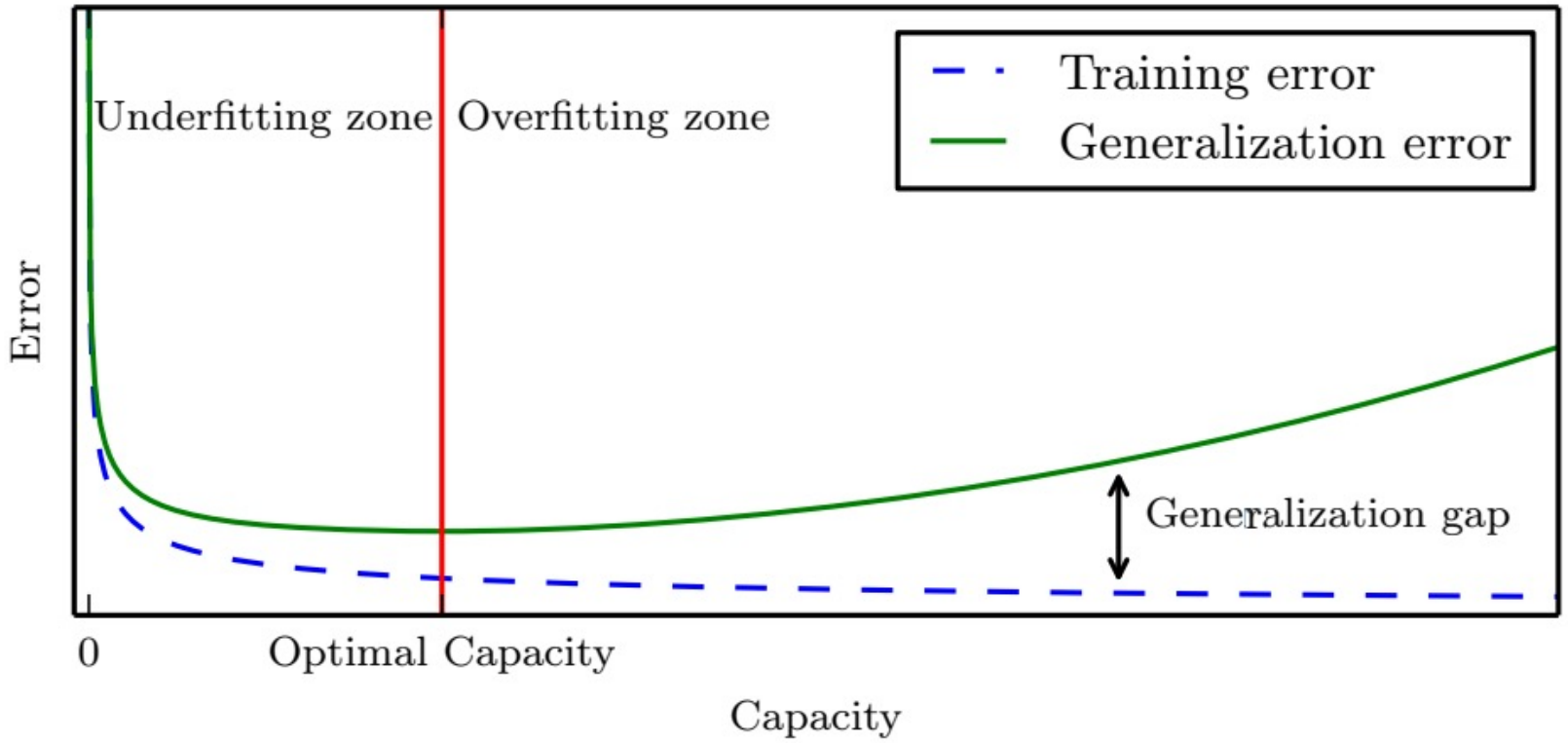
$L_p = \|\Sigma w_i\|_0 = \#w$

Goal: so try to bring down weight have max value



Goodfellow et al., 2016

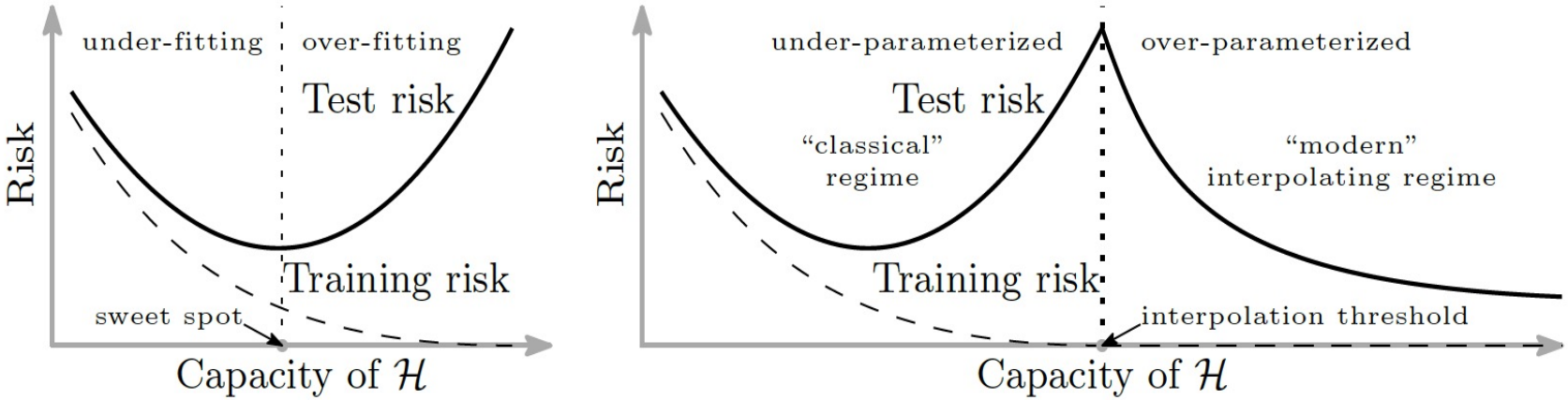
Underfit/Overfit Paradigm



Goodfellow et al., 2016

Conventional Wisdom Doesn’t work!

New “Interpolation” Regime



k-nearest neighbor classifier

an example of non-NN over-parameterized model

here increase # params is increase k

- No training required

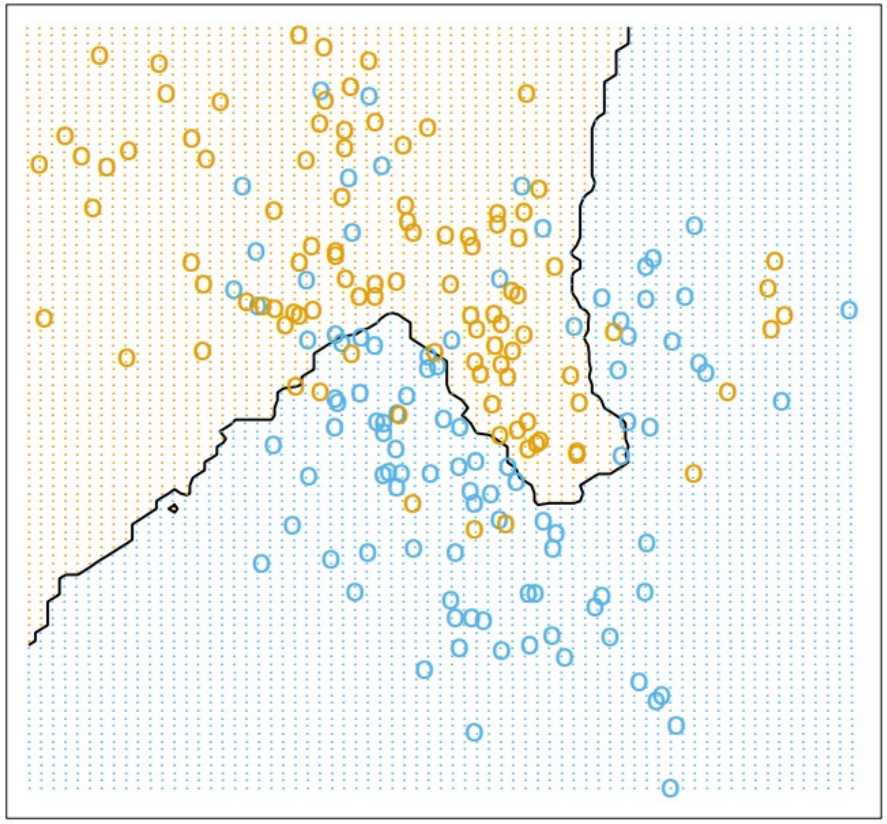
1. Find the k-nearest neighbors of the test point x in the training data
   - Call this set $\mathcal{N}_k(x)$
2. Choose $\hat{y}$ according to a majority vote of the corresponding labels of the neighbors in $\mathcal{N}_k(x)$

Interpolation Regime 插值机制

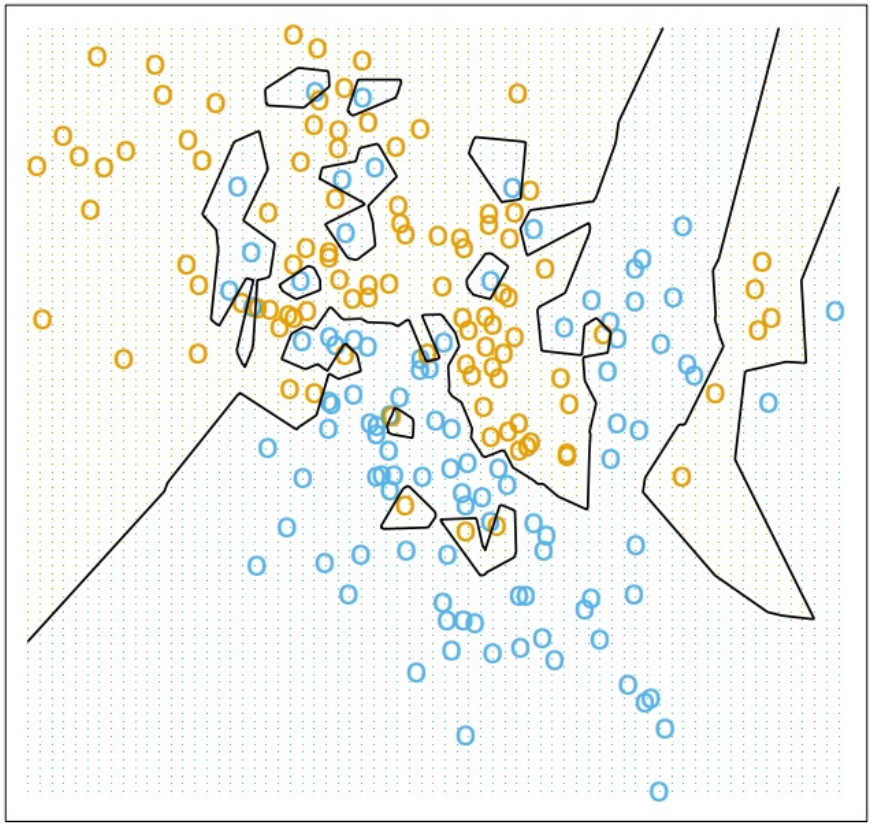
smoother

overfitting

15-Nearest Neighbor Classifier



1−Nearest Neighbor Classifier



k-nearest neighbor classifier

Hastie et al., 2009

Measuring test error

- Different from training error
- How do we select the data for estimating the two?
- Common approaches:
  - **Holdout validation**: Randomly split the data into two disjoint sets. Train the model on one set and compute test error on the other.
  - **Random subsampling**: Repeat the holdout validation k times independently and compute the mean test error
  - Typically these approaches are not very robust

Caveat on NN

first train { traing set : A, test set : B } ⇒ 2 different NNs with different weights and biases

second train { traing set : B, test set : A } hard to report p-value

Cross Validation

- Cross validation averages evaluation scores over several different splits to provide a more realistic assessment
- Types of cross validation
  - **Leave-one-out**: Use all data points except one for training and validate on the remaining point.
  - **2-fold cross validation**: Randomly split the data into two halves and run two validation iterations, each using one half for training and the other for validation
  - **k-fold cross validation**: Randomly partition the data into k equal size parts and run k validation iterations, each using one part for validation and the rest for training
- In all cases, the validation scores are averaged
- Each data point is only used once in testing but many times in training

another layer, randomization of training data

Further reading

- Chapter 5 in Goodfellow book
- *Elements of Statistical Learning* by Hastie, Tibshirani, & Friedman
- *Pattern Recognition and Machine Learning* by Bishop