Machine Learning

B. N. M. Institute of Technology

https://www.bnmit.org/

Department: Artificial Intelligence and Machine Learning (AI)

Course: Machine Learning (18AI61) effective from the academic year 2018–2019 under VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI

Instructor: Pradip Kumar Das

Refer companion code at https://github.com/PradipKumarDas/Teaching/tree/master/Machine_Learning_Course_18AI61

Machine Learning Landscape

What is Machine Learning?

Few Examples

Spam Email Filter

Recommender

Optical Character Recognition

Chatbots

Voice Recognition

Facial Recognition

Forecasting

Autonomous Vehicle

Medical Diagnostics

Fraud Detection

and many more...

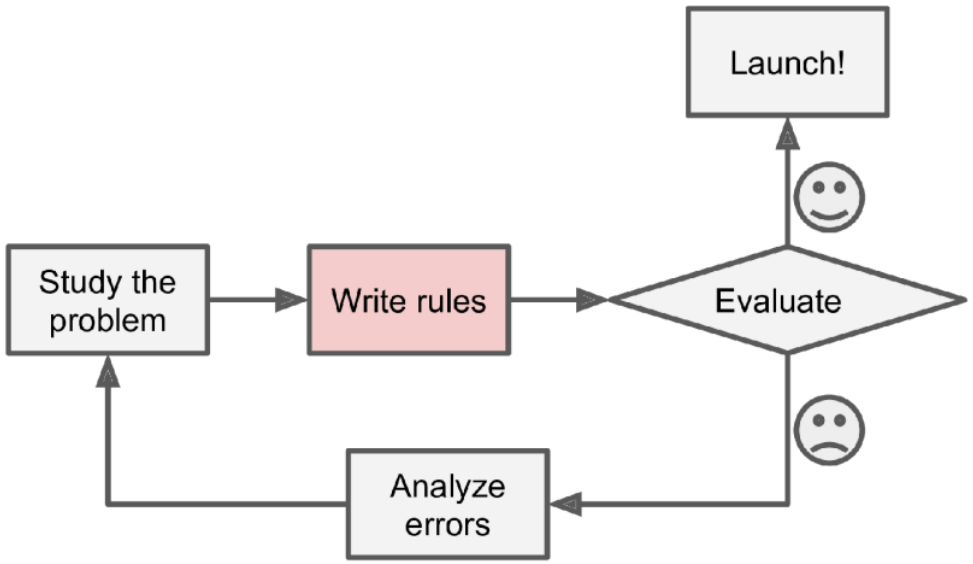
Some Definitions

"Science of programming computer to learn from data

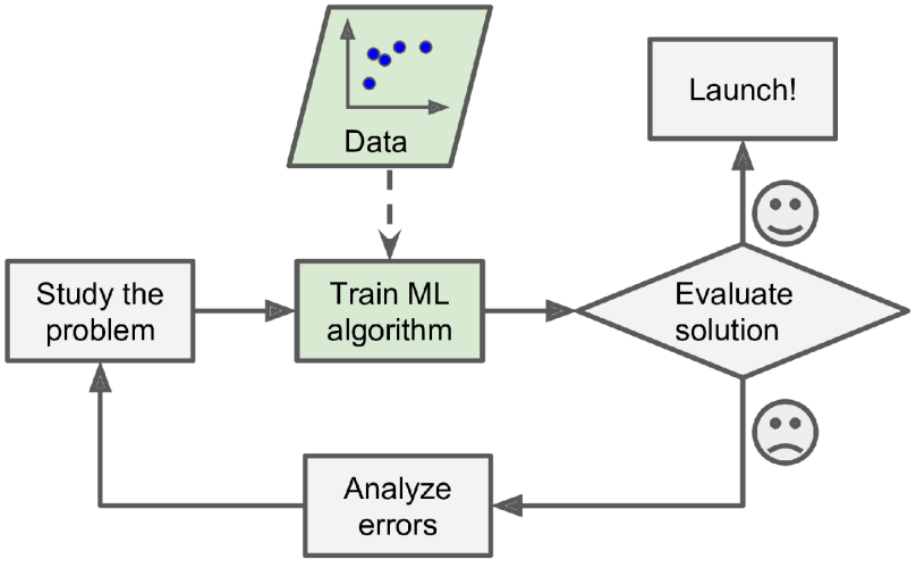
"field of study that gives computers the ability to learn without being explicitly programmed

*"A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E***

Why is Machine Learning Required?

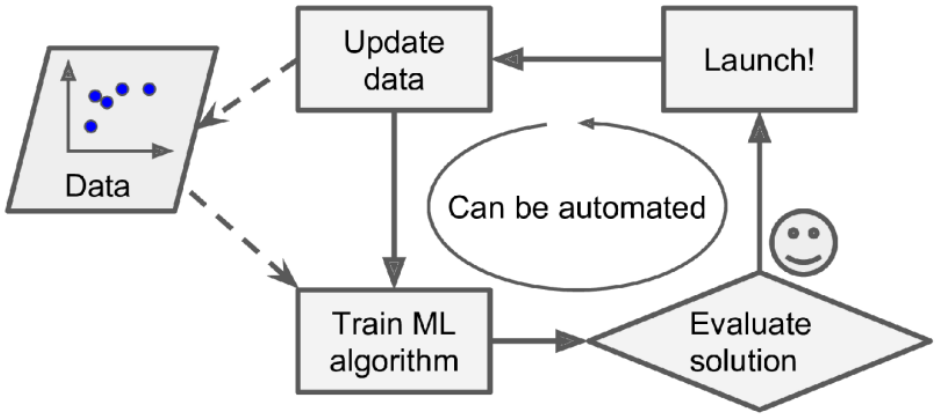


Traditional program writing approach

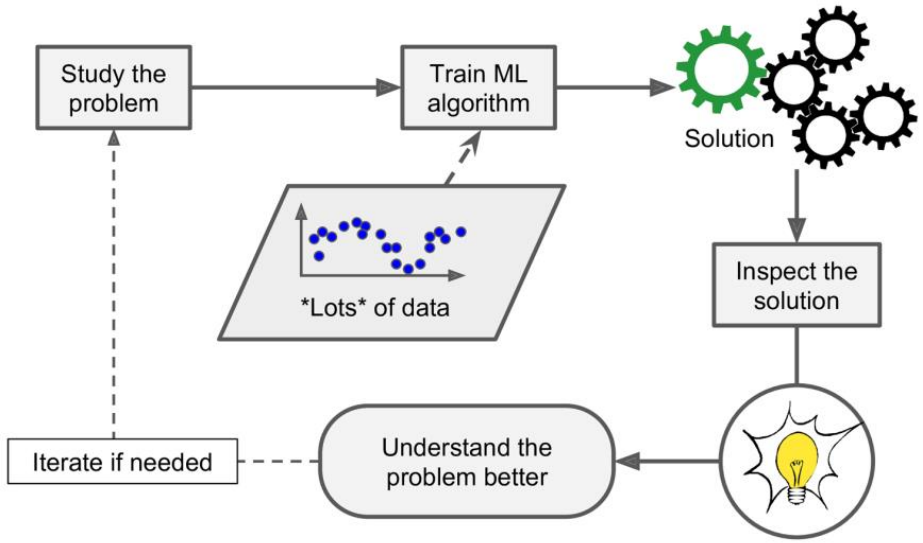


The Machine Learning approach

Why is Machine Learning Required? (Cont.)



Automatically adapting change



Discovering patterns for human learning

Machine Learning is a Good Option for

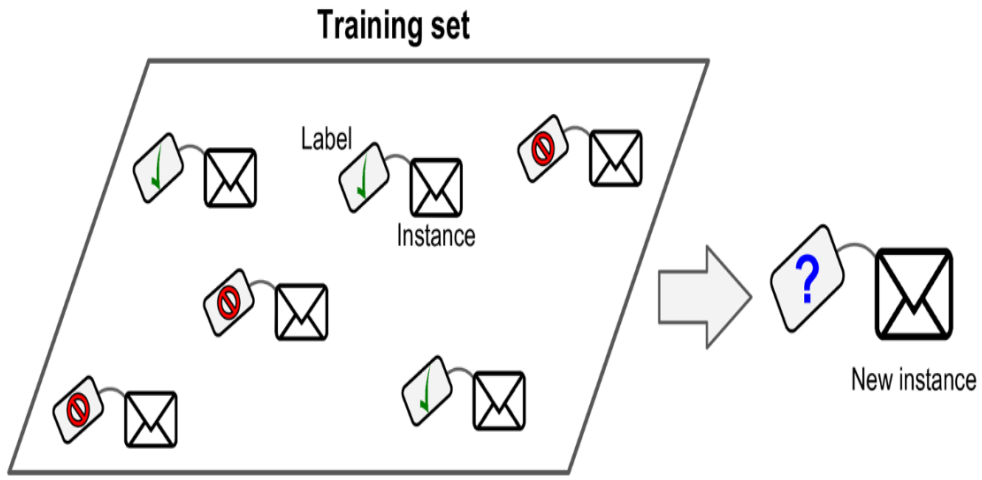
- Problems that require a lot of fine-tuning and long list of rules
- Problems that traditional approaches do not yield good results
- Problems where data distribution changes over time
- Getting insights about complex problems from large volume of data

Types of Machine Learning Systems

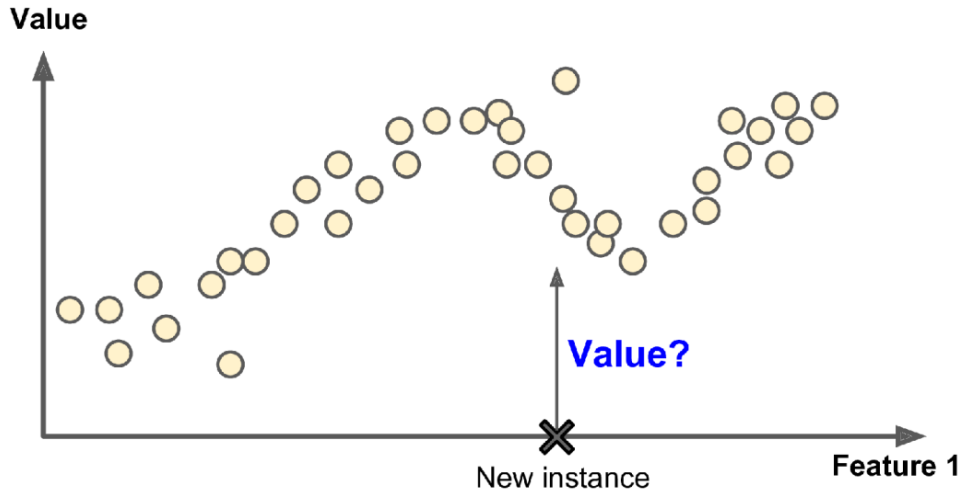
Whether learning requires human supervision	Whether learning is incremental	Whether learning is to compare or to detect pattern and/or predict
• Supervised Learning • Semi-supervised Systems • Unsupervised Learning • Reinforcement Learning	• Batch Learning • Online Learning	• Instance-based Learning • Model-based Learning

Types of Machine Learning Systems (Cont.)

Classification & Regression in Supervised Learning



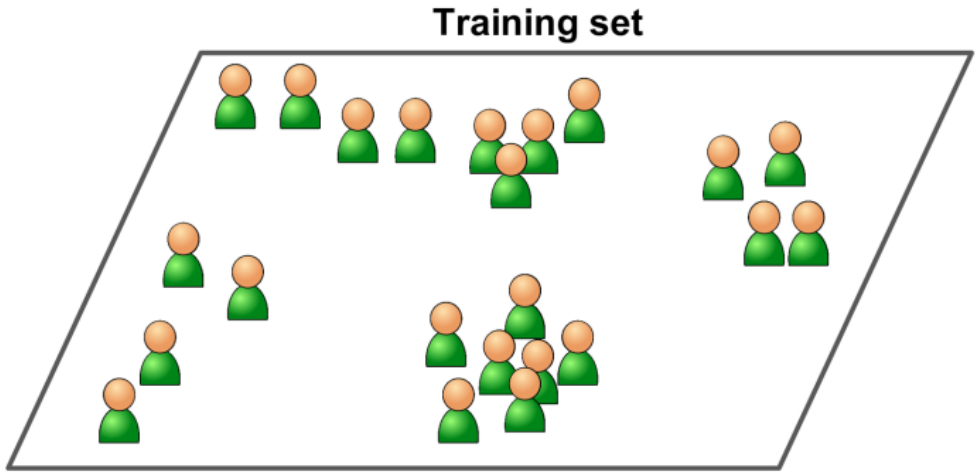
A labeled training set for spam classification



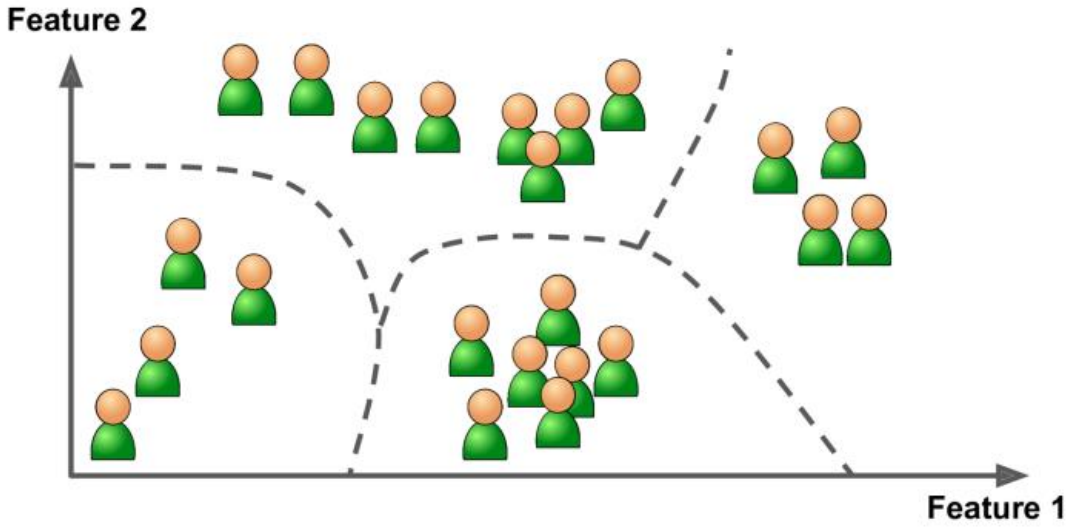
Predicting a continuous value in regression problem

Types of Machine Learning Systems (Cont.)

Clustering in Unsupervised Learning



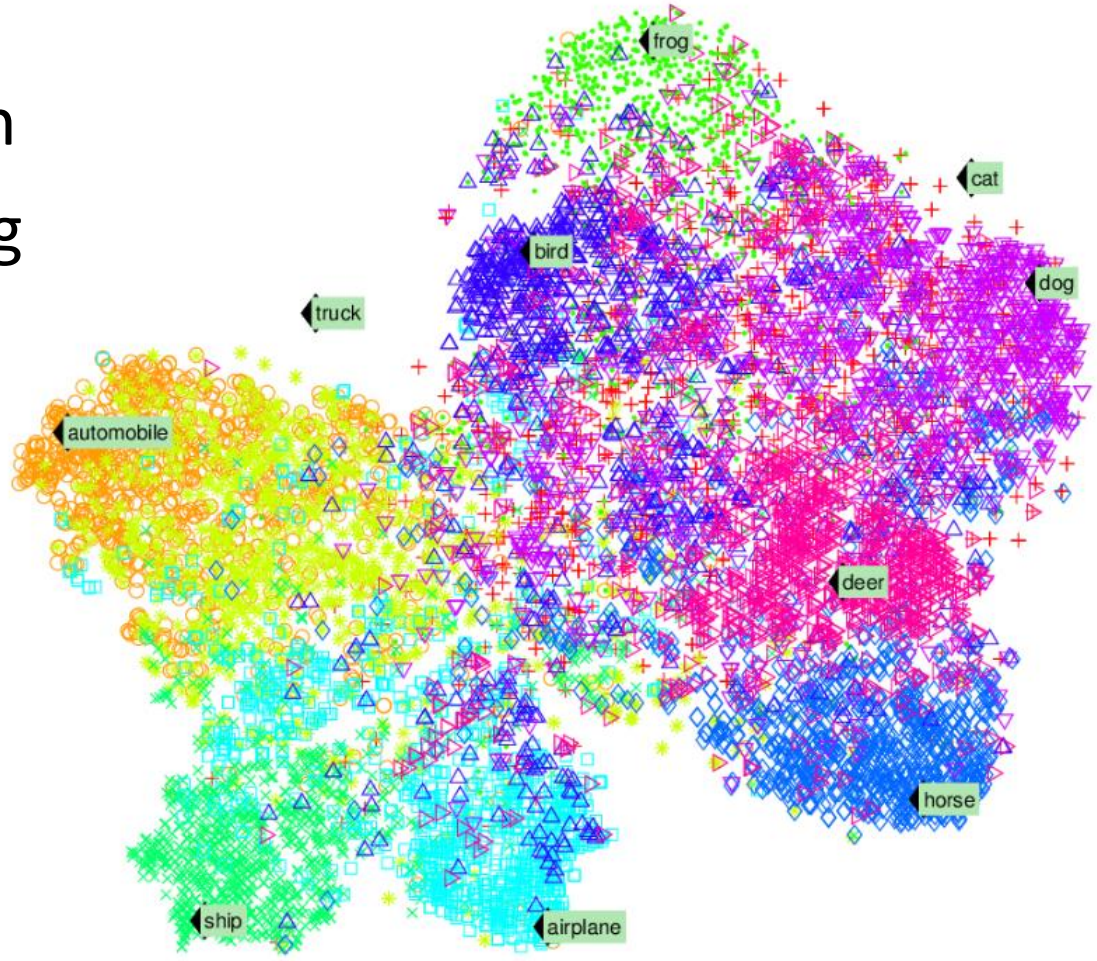
An unlabeled training set for clustering



Detecting groups of similar visitor in clustering

Types of Machine Learning Systems (Cont.)

Cluster Visualization in Unsupervised Learning



Cluster visualization in unsupervised learning

Types of Machine Learning Systems (Cont.)

Dimensionality Reduction in Unsupervised Learning

- Simplifying data without losing too much information
- Merging correlated features into one
- Feature extraction

Types of Machine Learning Systems (Cont.)

Anomaly and Novelty Detection in Unsupervised Learning

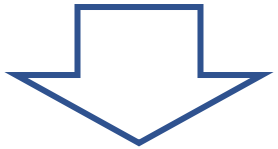
Detection of an anomaly

Types of Machine Learning Systems (Cont.)

Association Rule Learning in Unsupervised Learning

{Potatoes, Onions} => {Burger}

{Barbeque sauce, Potato chips} => {Steak}

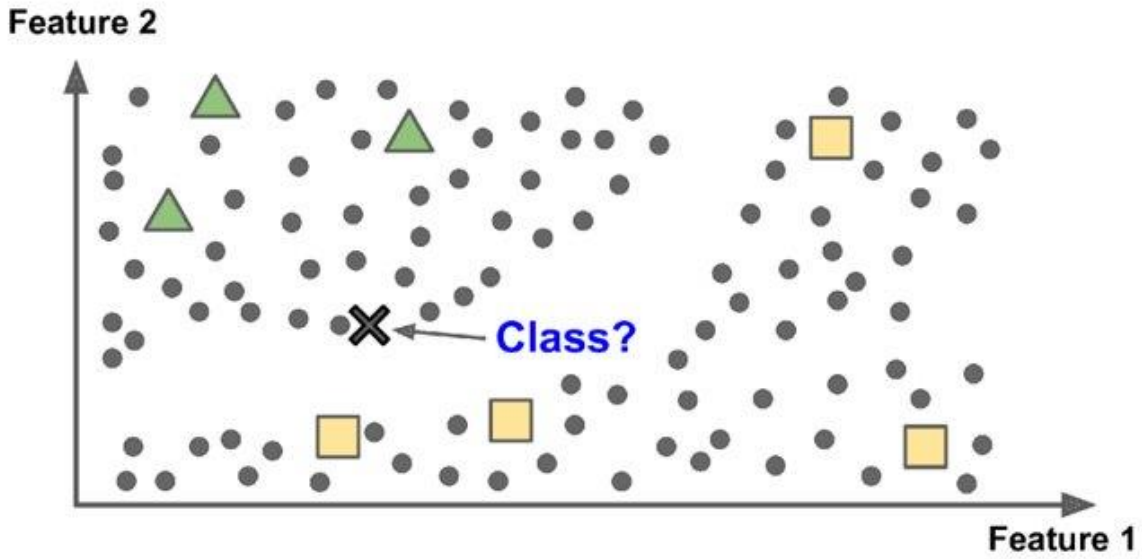


Results in Promotional Pricing and/or Appropriate Product Placement

Sales logs revealing customers' buying pattern

Types of Machine Learning Systems (Cont.)

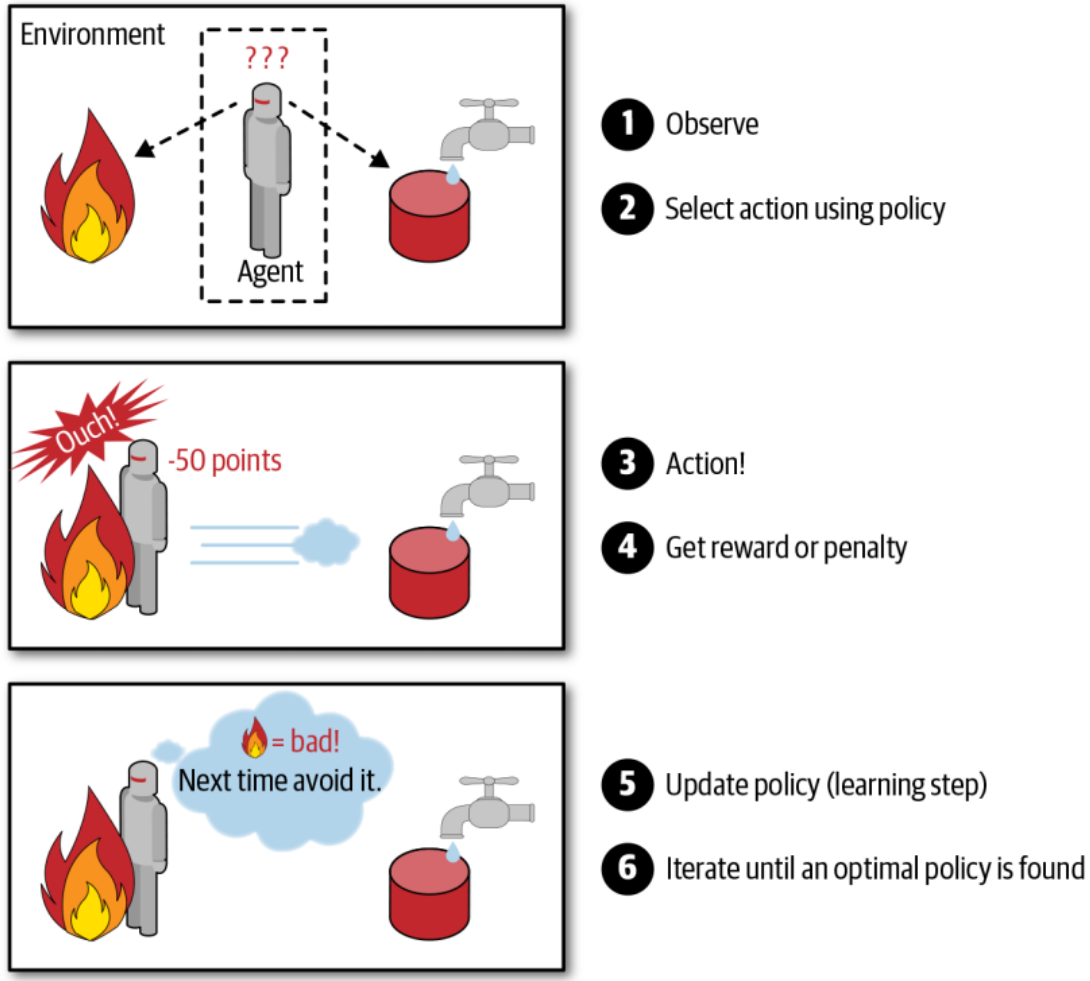
Classification in Semisupervised Learning



Few labeled examples help classifying new instances

Types of Machine Learning Systems (Cont.)

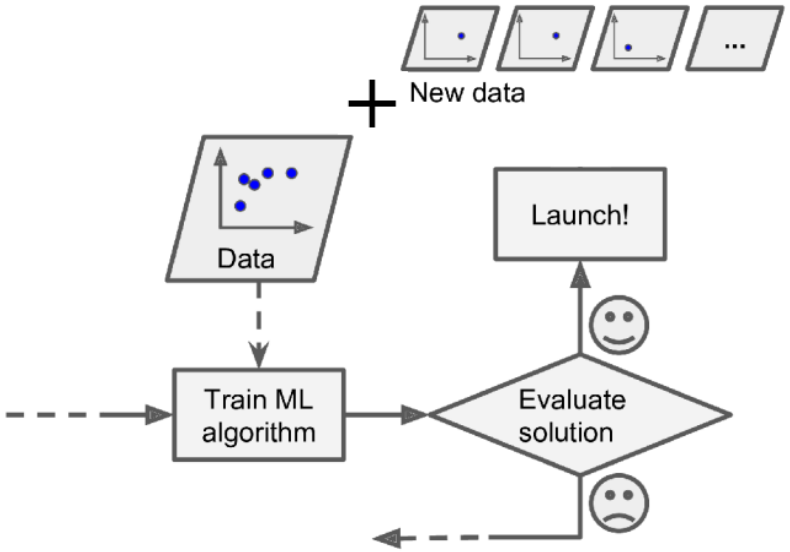
Building Strategy in Reinforcement Learning



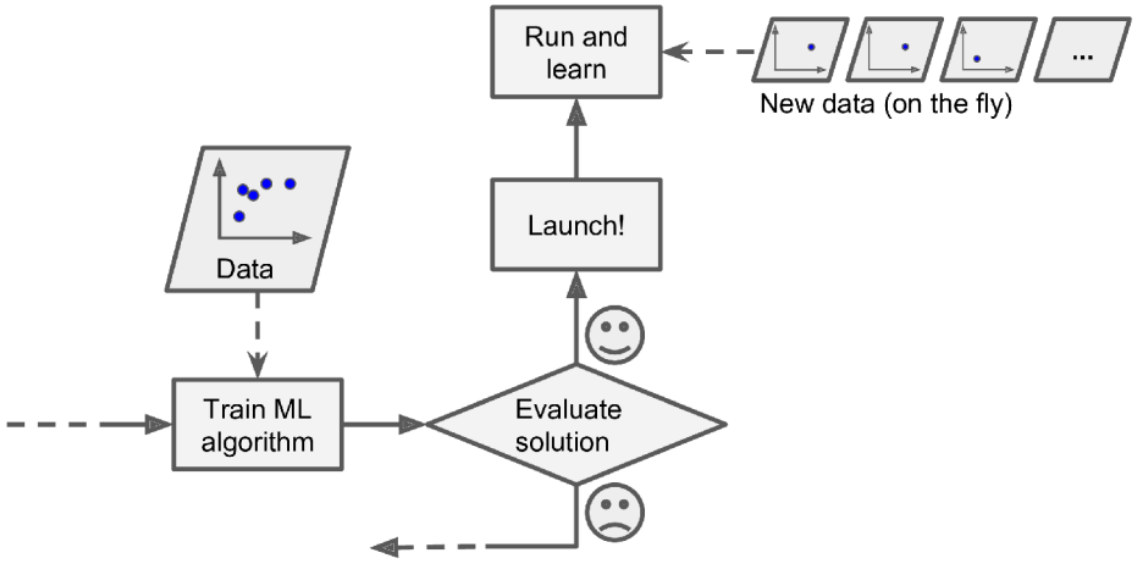
Receiving reward or penalty against its actions and fine-tuning its strategy

Types of Machine Learning Systems (Cont.)

Batch and Online Learning

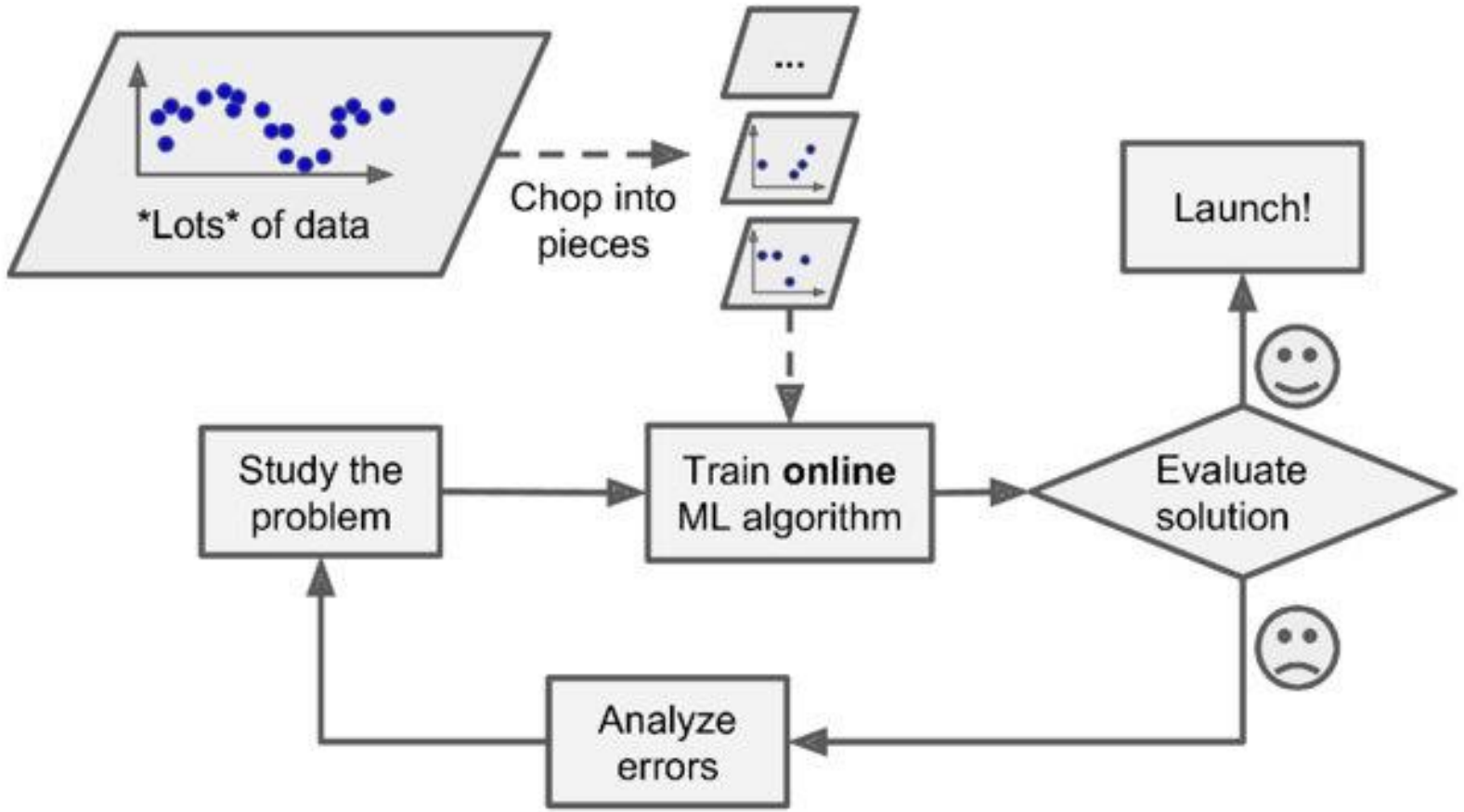


Relearning from scratch on full dataset (old and new data) in Batch Learning



Learning is incremental for new data in Online Learning

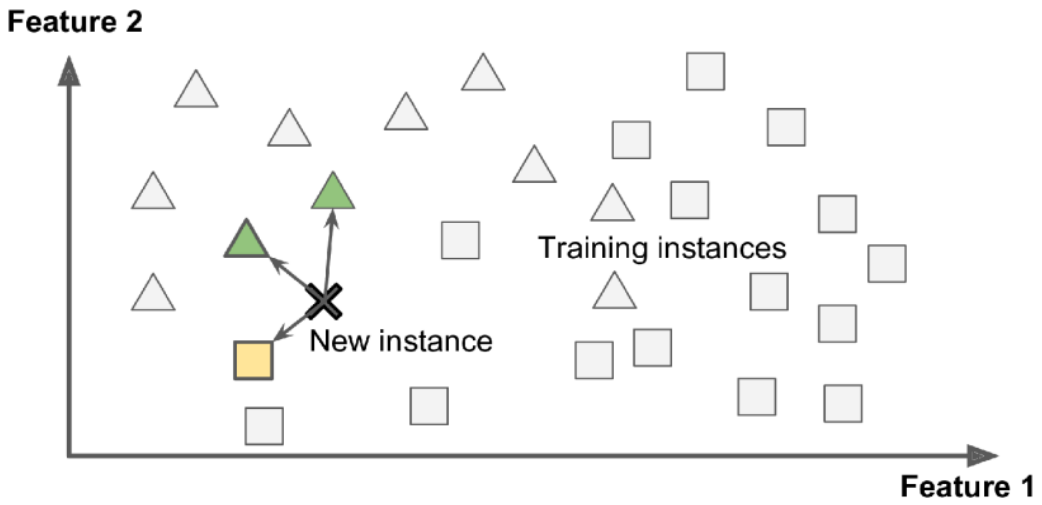
Types of Machine Learning Systems (Cont.)



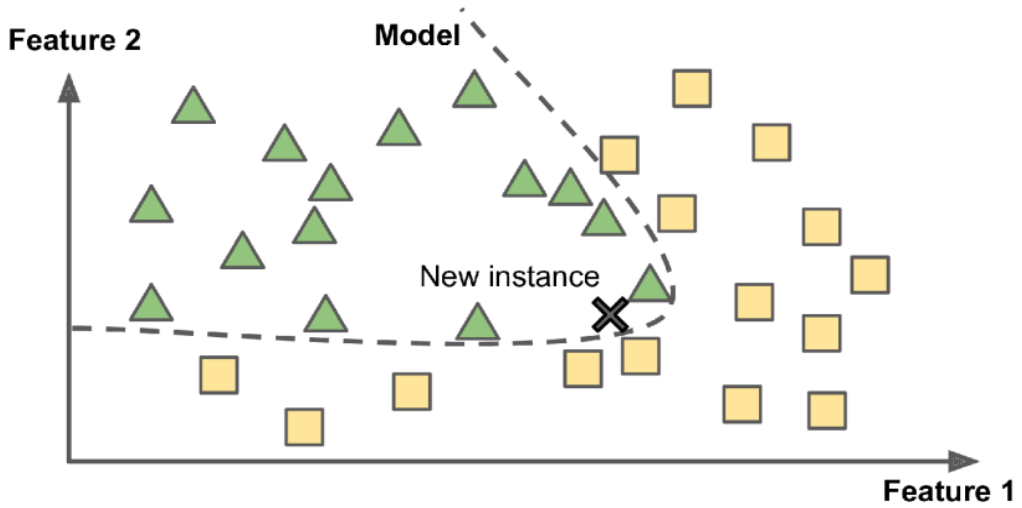
Handling huge dataset in Online Learning

Types of Machine Learning Systems (Cont.)

Instance-based or Model-based Learning systems on how they generalized



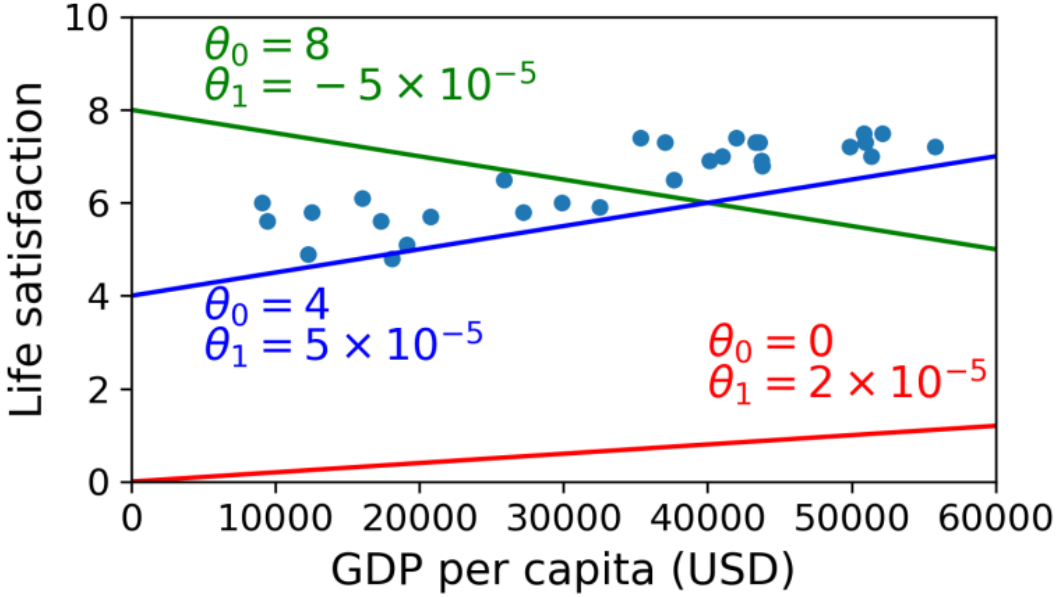
Instance-based Learning is by heart and it then generalizes to new cases by using similarity measures



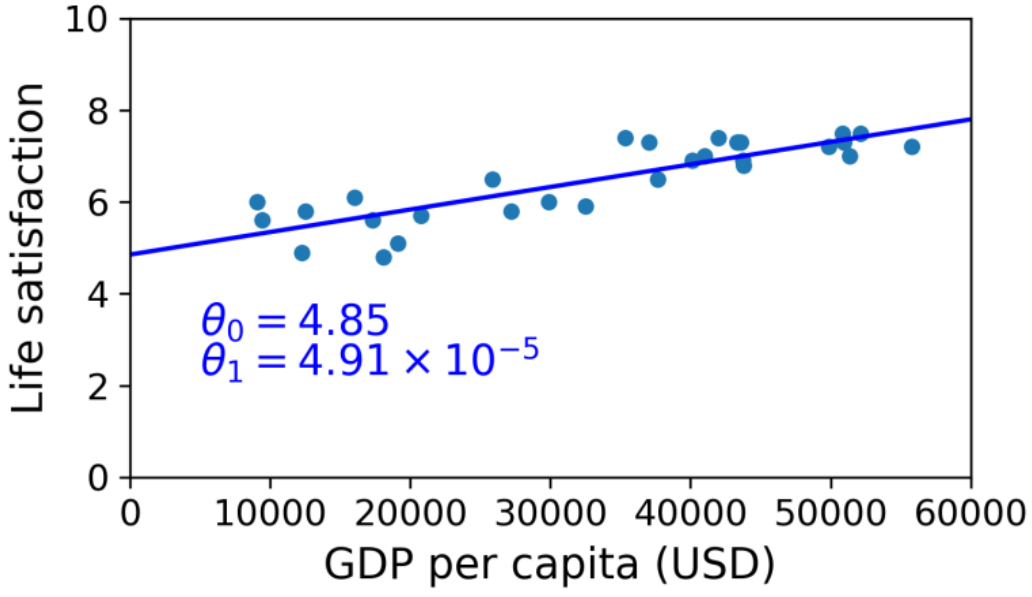
Model is created in Model-based Learning and then it make predictions

Types of Machine Learning Systems (Cont.)

Fitting Model to Data in Model-based Learning



Possible linear models

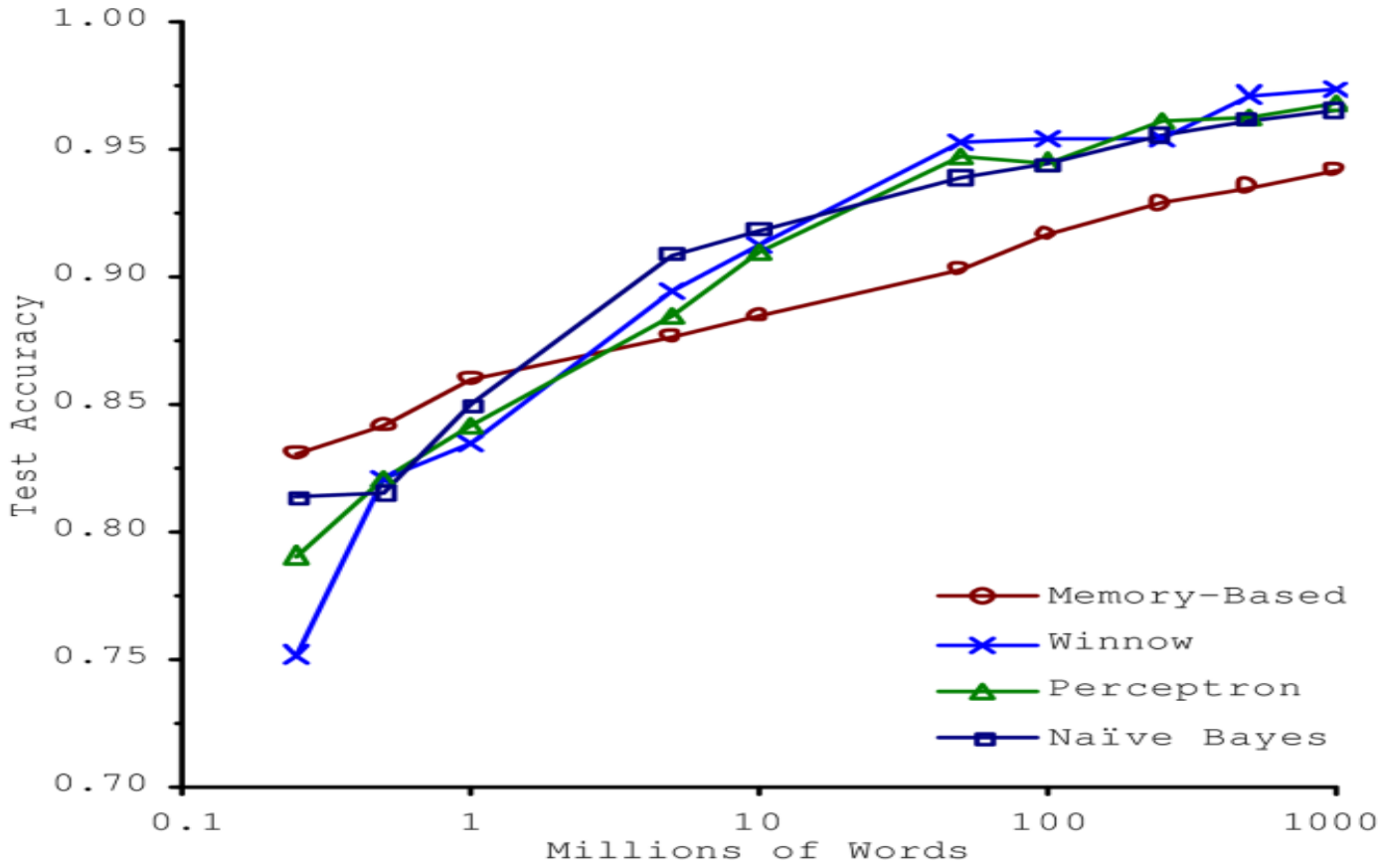


Fitted linear model

Main Challenges in Machine Learning

Insufficient Quantity of Training Data

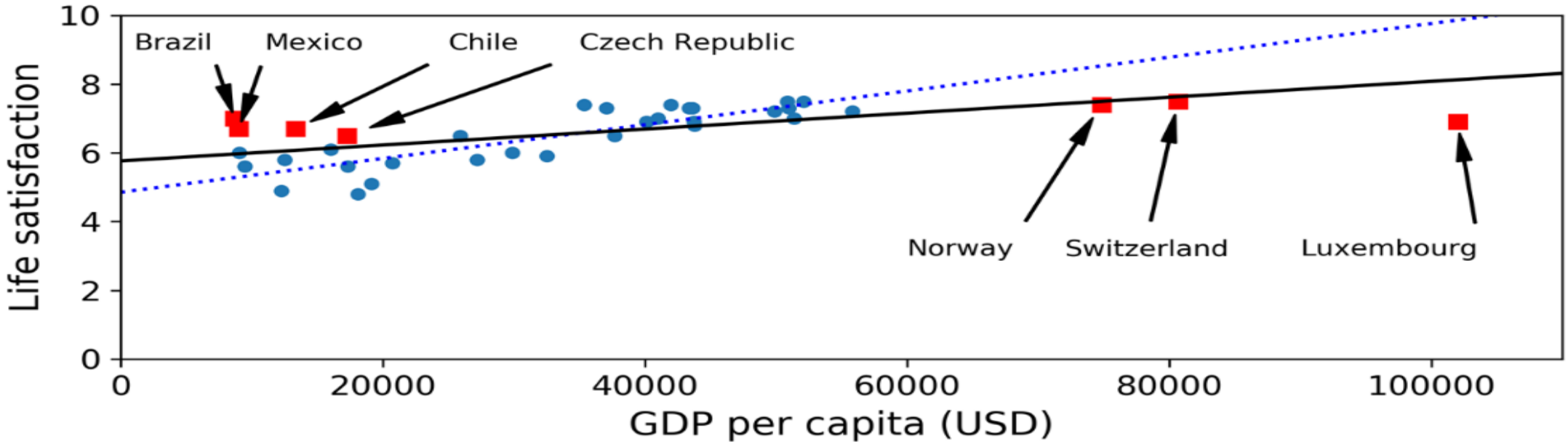
Trade-off between spending time and money on algorithm development and spending these on corpus development



Main Challenges in Machine Learning (Cont.)

Nonrepresentative Training Data

- Sampling Noise
- Sampling Bias



A better model fitted over more representative training data

Main Challenges in Machine Learning (Cont.)

Poor-Quality Data

- Errors
- Missing values
- Outliers

Main Challenges in Machine Learning (Cont.)

Irrelevant Features

Applying Feature Engineering involving the following steps

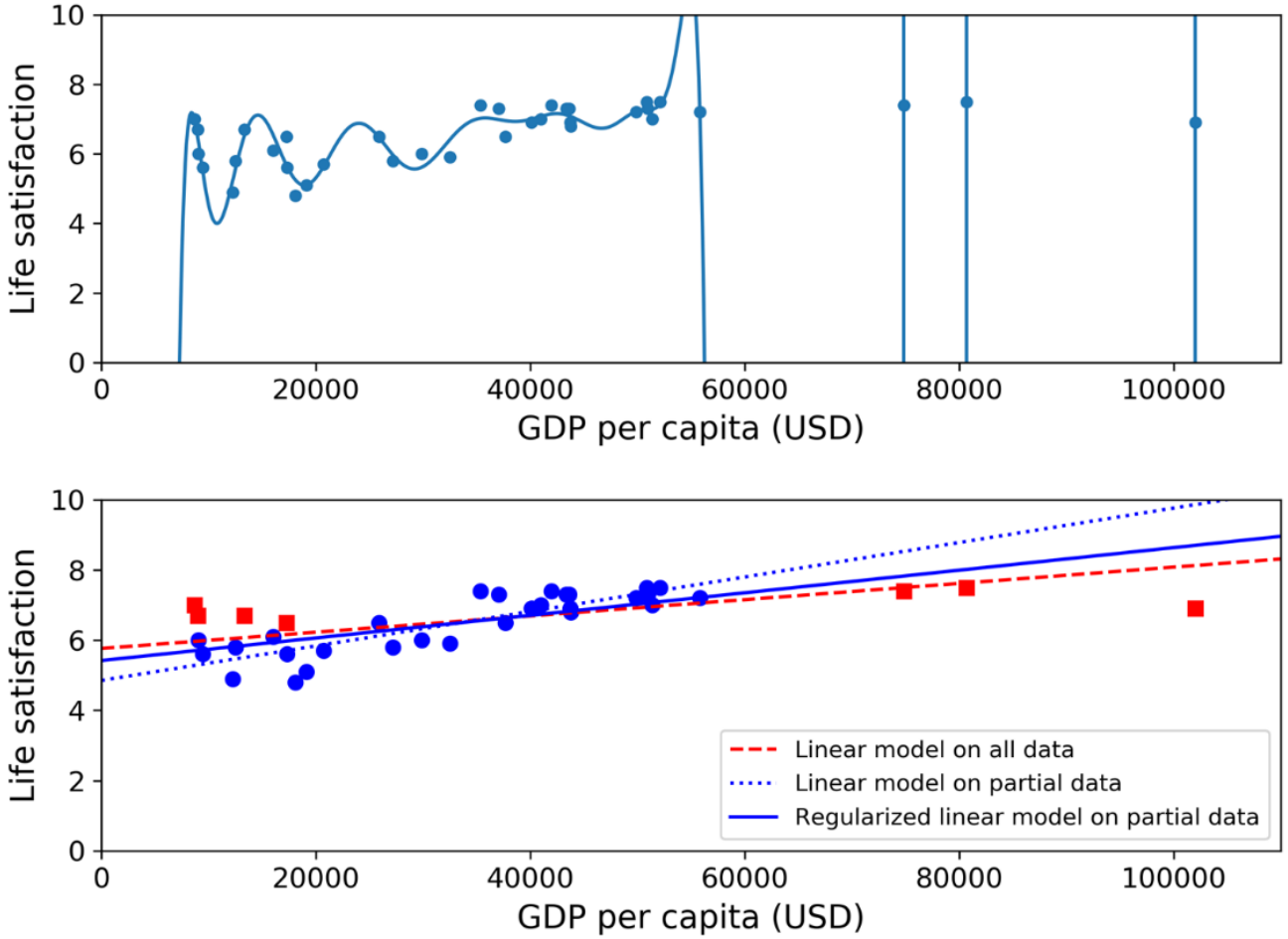
- Feature selection
- Feature extraction
- Creating new features

Main Challenges in Machine Learning (Cont.)

Overfitting the Training Data

Resolving by

- Simplifying model
- Applying constraints (regularization)
- Gathering more training data
- Fixing data quality issues



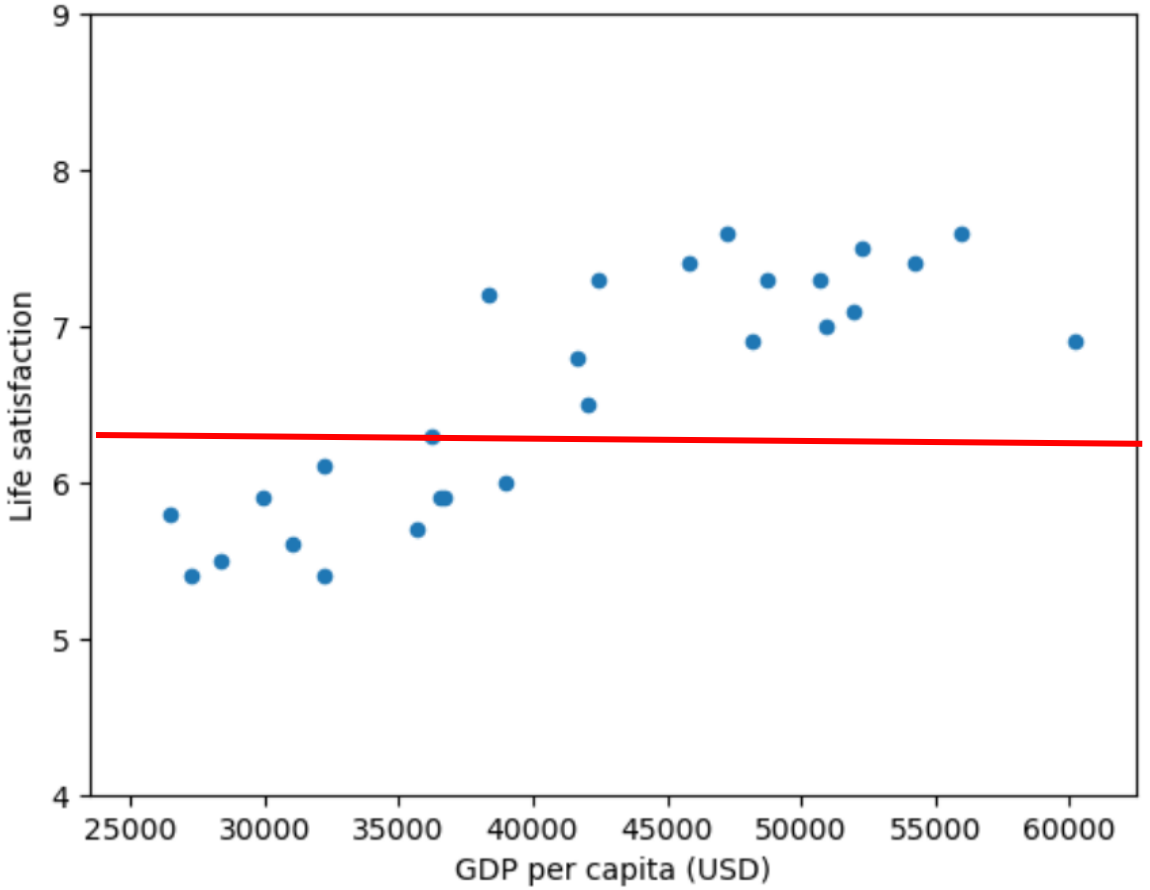
Overfitting and applying regularization to avoid it

Main Challenges in Machine Learning (Cont.)

Underfitting the Training Data

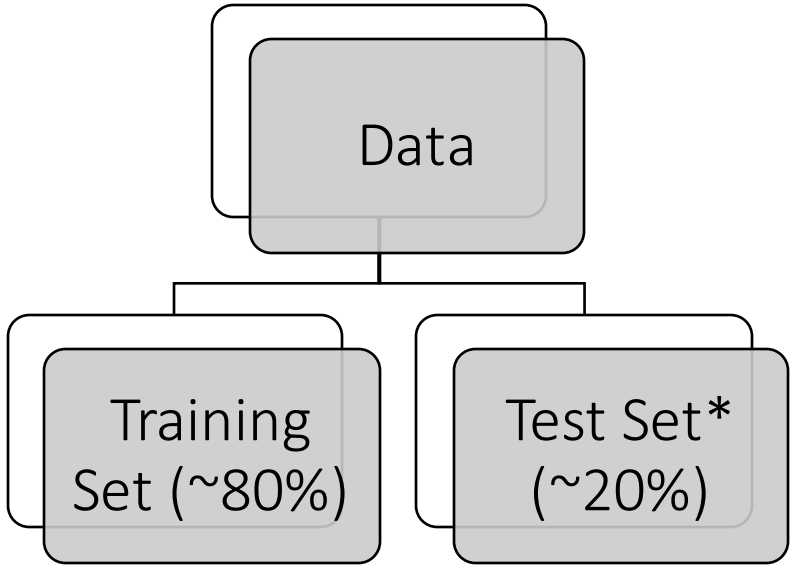
Resolving by

- Selecting more powerful model
- Feeding better features
- Reducing constraints or regularization



Testing and Validating

Ensuring Model Generalizes Well



1

Model is built by fitting it on the training data

2

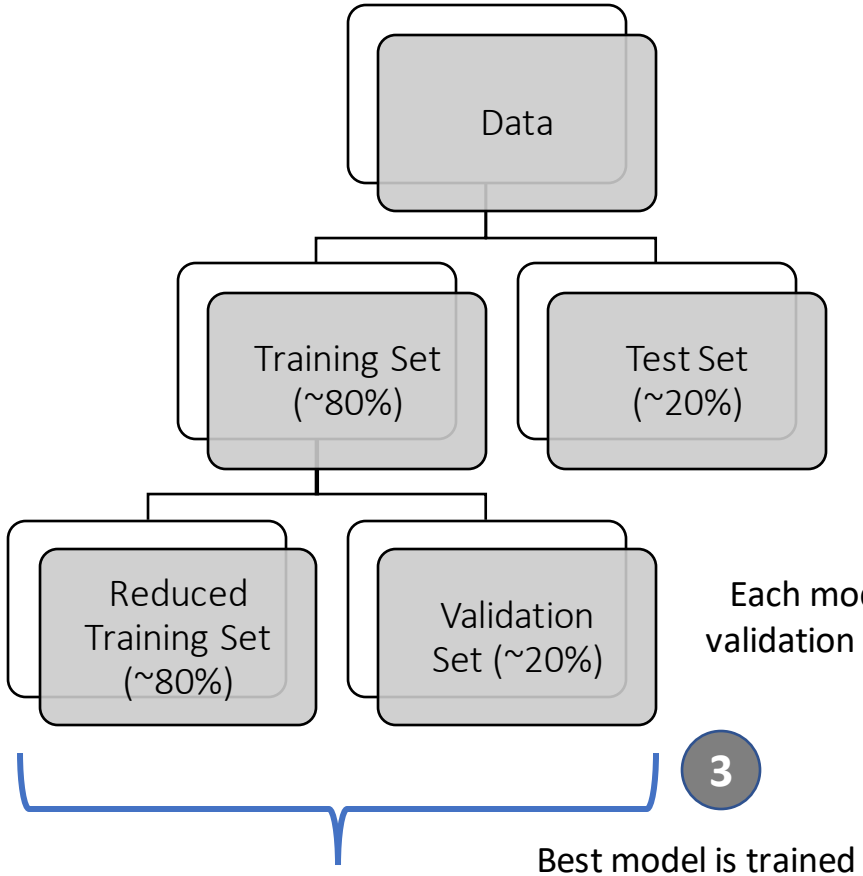
Test data helps estimating how the model generalizes

a.k.a. Hold-out set

A better model fitted over more representative training data

Testing and Validating (Cont.)

- Hyperparameter Tuning
- Model Selection
- Validation
- Cross-validation



1

Multiple models with different hyperparameters are created on reduced training data set

2

Each models gets tested on validation or hold-out data set

3

Best model is trained on whole training data set

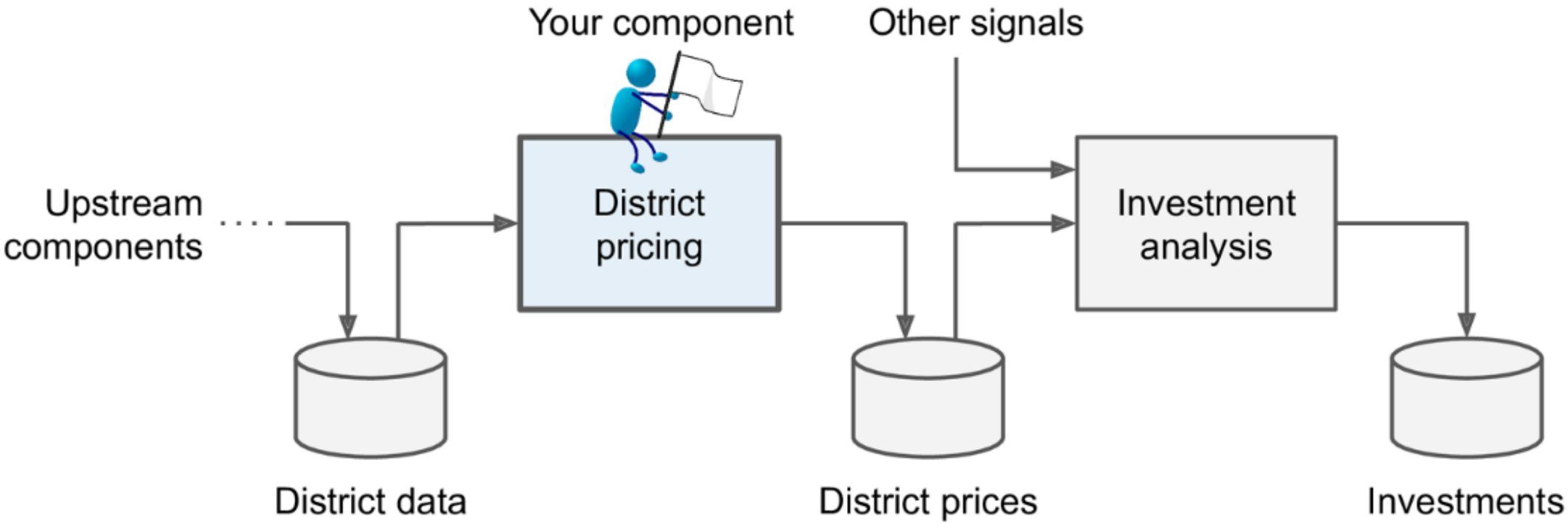
4

Final model gets tested on test set to estimate generalization error

No Free Lunch (NFL) Theorem

- No model that is *a priori* guarantees to be a better one
- Evaluating all models is only way to know which one works best – *Not practical*
- Hence, making *reasonable assumption about data* and *evaluating few reasonable models* is better option

The Pipeline



High-level machine learning pipeline for real estate investment

Defining Learning Problems & Designing a Learning System

Defining Learning Problem

Refer the following Machine Learning definition once again.

*"A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E***

To have a well-defined learning problem, the followings are to be identified

- The Task (***T***)
- The Performance Measure (***P***), and
- Training Experience (***E***)

Defining Learning Problem (Cont.)

Examples:

Learning Problem	Task (T)	Performance Measure (P)	Training Experience (E)
A checkers learning problem	Playing checkers	Percentage of games won against opponents	Playing practice games against itself
A handwriting recognition learning problem	Recognizing and classifying handwritten words within image	Percentage of words correctly classified	Database of handwritten words with given classification
A self-driving vehicle learning problem	Driving on one or multi-lane road	Average distance travelled before error occurs	Sequence of image and steering commands

Designing a Learning System

(Understanding through Checker game examples)

A. Choosing Training Experience
B. Choosing Target Function
C. Choosing Target Function Representation
D. Choosing a Function Approximation Algorithm
E. The Final Design

Designing a Learning System
(considering Checker game examples here)

Choosing Training Experience

a) Attributes of Training Experience Learning

1. Learning from feedback
   - *Direct* feedback
   - *Indirect* feedback
     - *Credit Assignment*
   - Learning from *Direct* feedback is earlier than learning from *Indirect* feedback
2. Controlling sequence of training examples
   - Teacher to provide board states with correct move for each
   - Learner to propose confusing board states asking teacher for the correct move
   - Learner experiments with novel board state (or minor variation of line of play) with indirect feedback while playing against itself
3. Distribution of training examples

Designing a Learning System (Cont.)

Choosing Training Experience (Cont.)

b) Defining learning problem

- Task *T*: Playing Checker
- Performance measure *P*: Percent of games won
- Training experience *E*: Games played against itself

Designing a Learning System (Cont.)

Choosing Target Function

- Determining *type of knowledge* to learn such that it can used in performance measurement (generating *legal* moves from any board state – the *Optimization* problem)
- Options for *target* function to learn

Target Function	Description	Comments
ChooseMove: B --> M	B is set of legal moves and M is chosen output move	Difficulty in learning from indirect feedback
V : B --> $\mathbb{R}$	Mapping any legal board state from the set of B to some real value	Easier to learn to assign a numerical score to each any board state

Designing a Learning System (Cont.)

Choosing Target Function (Cont.)

V : B --> $\mathbb{R}$

Assigning higher score to better board state

Condition	*V(b)* Output
***b* is final board state that is won**	100
***b* is final board state that is lost**	-100
***b* is final board state that is drawn**	0
***b* is not a final board state**	*V(b')* where *b'* is the best board state that can be achieved starting from *b* and playing optimally till the end of game

V(b)=V(b') is a *non-operational* definition as it is not efficiently computable. Instead, an *operational* version of *V^ (read V-hat)* is to be discovered and *approximated*, and this process is also called *Function Approximation*.

Designing a Learning System (Cont.)

Choosing Target Function Representation

The Options:

- A large table with a distinct entry specifying the value for each distinct board state
- A collection of rules that match against features of the board state
- A quadratic polynomial function of predefined board features
- An artificial neural network

An Example $V\hat{}$ Representation:

$$V\hat{}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

w_0 through w_6: weights or coefficients
x_1: No. of black pieces
x_2: No. red pieces
x_3: No. of black kings
x_4: No. of red kings
x_5: No. of black pieces threatened by red
x_6: No. of red pieces threatened by black

Designing a Learning System (Cont.)

Choosing a Function Approximation Algorithm

- Training examples each describing a specific board state *b* and training value for *b* in the form of <*b*, $V_{train}(b)$> are required to learn approximation function V^

Example:

<<x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0>, +100> where black has won the game

- Estimating training values
  - It's easy to assign a value to a board state that corresponds to the end of game
  - It's difficult to assign values to all intermediate board states before game ends
  - Rule for estimating training values: $V_{train}(b)$ <-- *V^(Successor(b))*

Designing a Learning System (Cont.)

Choosing a Function Approximation Algorithm (Cont.)

- Adjusting the weights
  - Learning algorithm to choose values for weights w_i to best fit training examples $\{\langle b, V_{train}(b)\rangle\}$
  - Minimizing squared error E using learning algorithm such as Least Mean Square (LMS)

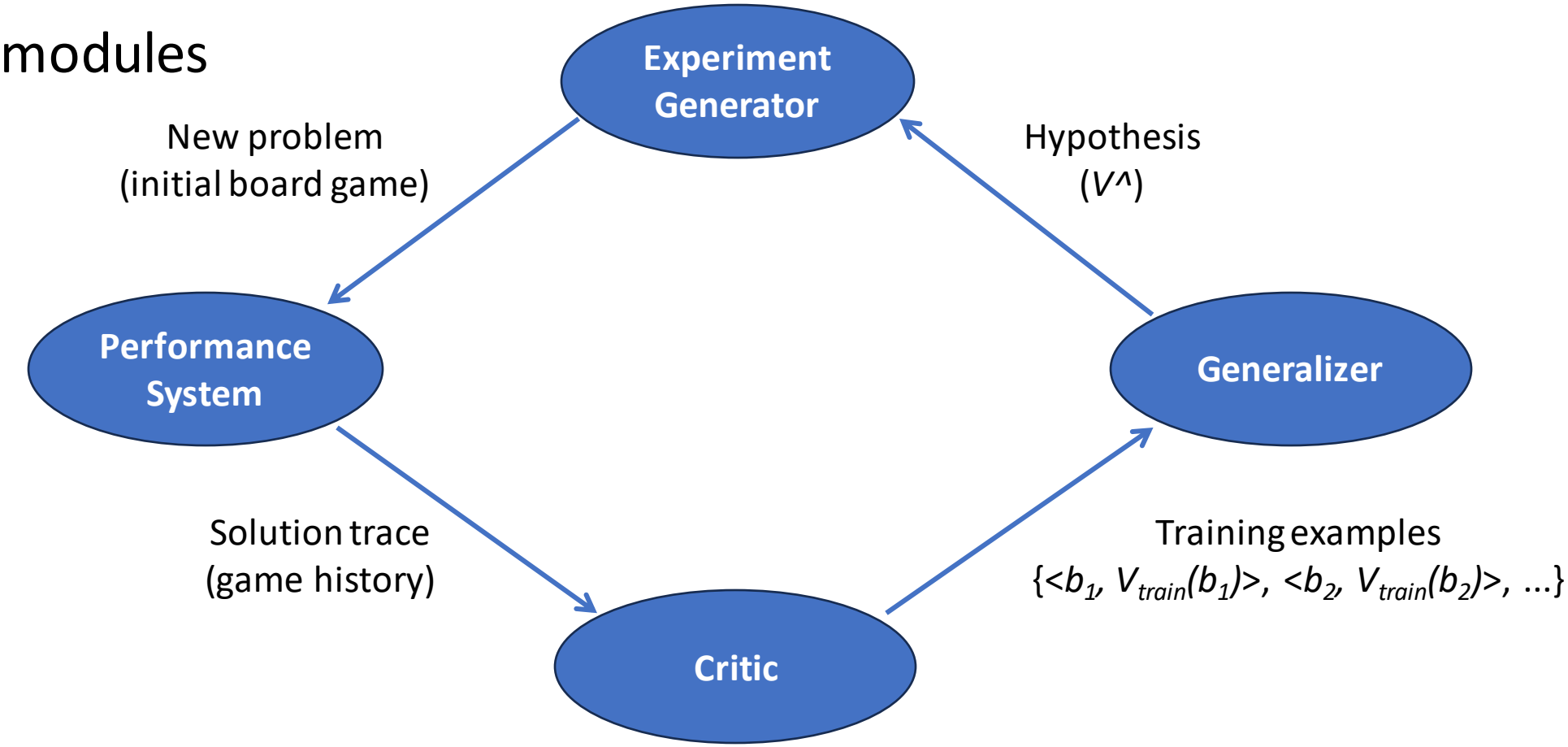
$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{training examples}} (V_{train}(b) - \hat{V}(b))^2$$

  - LMS weight update rule
    - For each training example $\langle b, V_{train}(b)\rangle$
      - Use the current weight to calculate $V\hat{}(b)$
      - For each weight w_i, update it as $w_i \leftarrow w_i + \eta(V_{train}(b) - V\hat{}(b))x_i$

Designing a Learning System (Cont.)

The Final Design

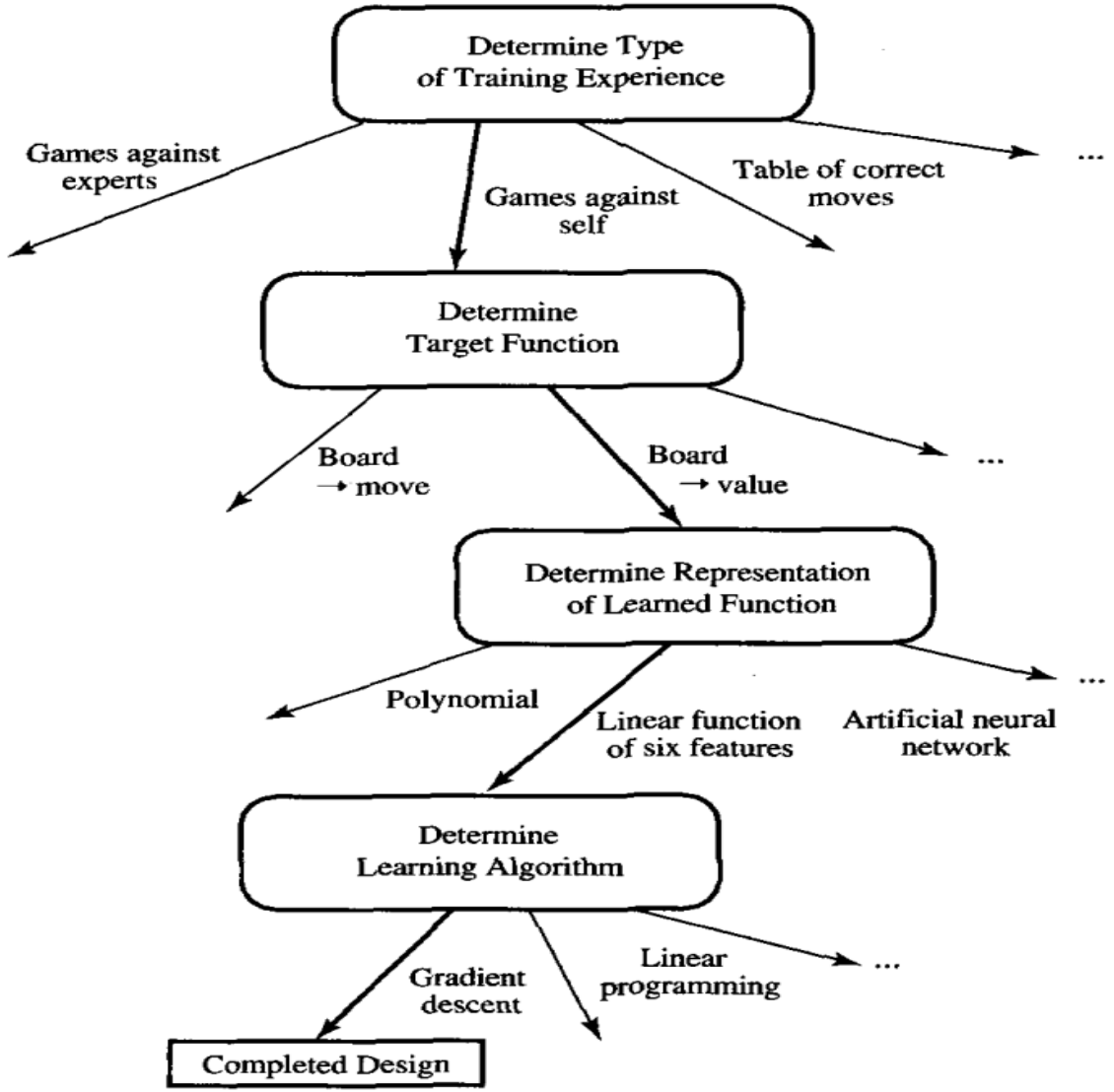
Four core modules



Designing a Learning System (Cont.)

The Final Design (Cont.)

The (sequence of) design choices



Perspectives on Machine Learning

- Involves searching a large space of possible hypotheses to determine one that best fits the observed data
  - for checker, hypothesis space consist of evaluation functions that can be represented by some choices of values for the weights w_0 through w_6
- Algorithms that searches through hypothesis space
  - Linear Functions
  - Decision Trees
  - Artificial Neural Network, etc.
- Relationship between the size of the hypothesis space to be searched and number of training examples available, and the confidence that a hypothesis consistent with the training data will correctly generalize to unseen examples

Issues in Machine Learning

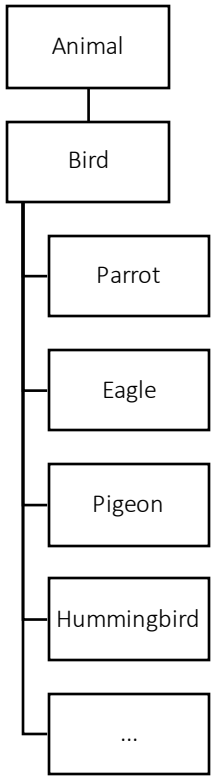
- Algorithms
  - What algorithms exist for learning general target function from specific examples?
  - In what settings will a particular algorithm converge to the desired function, given sufficient training data?
  - Which algorithms perform best for which type of problems and representations?
- Training data
  - How much training data is sufficient?
- Learning function
  - What's the best way to reduce learning task to one or more function approximation problems?
  - How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Concept Learning

Automatically inferring the general definition of some concept

Introduction

- Learning general concepts such as "bird"
- Each concept can be viewed as describing some subset of objects or events defined over a larger set



Definition

Inferring a boolean-valued function from training examples of its input and output

Concept Learning Task

- **Learning from Examples:** *Days on which my friend Aldo enjoys his favorite water sport*

The Instances (X)

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- Hypothesis Representation: *A Vector of 'n' constraints* h = <| | | | | | |>
- Each constraint will either be '?' (any value is acceptable), any single value from respective example attributes, or 'Φ' (no value is acceptable)
- ***h(x) = 1:*** *'h'* classifies *x* is positive examples provided *x* satisfies all constraints of hypothesis *h*

Concept Learning Task (Cont.)

- Example Hypotheses:

Hypothesis	Interpretation
<?, ?, ?, ?, ?, ?>	Every day is a positive day
<?, Cold, High, ?, ?, ?, ?>	Only cold days with high himidity are positive days
<Φ, Φ, Φ, Φ, Φ, Φ>	No day is a positive day

- *c: X* --> {0, 1} where c is boolean-valued function for concept, *X* represents instances and {0, 1} is boolean output

General-to-Specific Ordering of Hypotheses

- More general then...

  h1 = <*Sunny*, ?, ?, *Strong*, ?, ?>

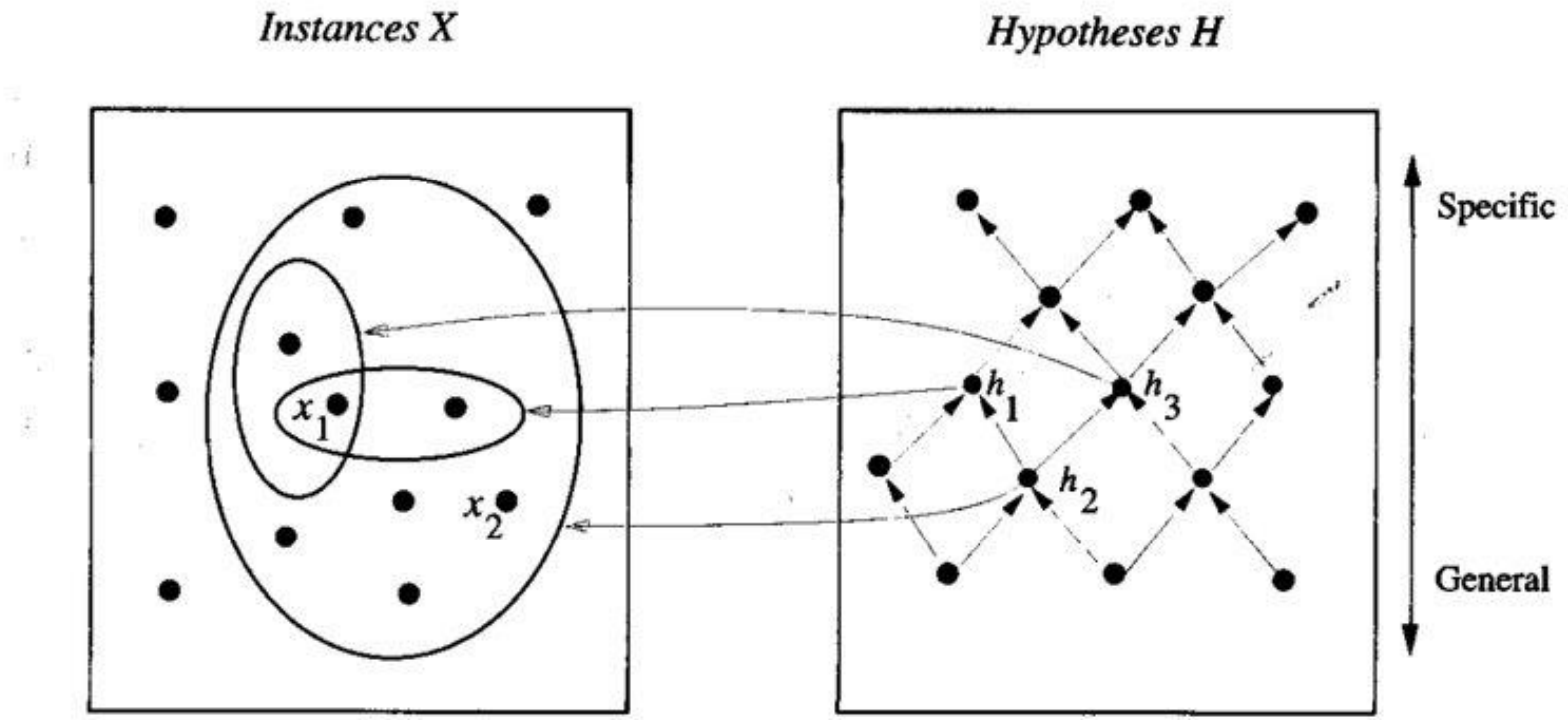
  h2 = <*Sunny*, ?, ?, *?*, ?, ?>

  h2 classifies more instances as positive, and hence *h2* is more general than *h1*

- Let h_j and h_k to be boolean-valued functions defined over X. Then h_j is more general than or equal to h_k if and only if

  $(\forall x \in X)[h_k(x) = 1) \text{ --> } (h_j(x) = 1)]$

General-to-Specific Ordering of Hypotheses (Cont.)



x_1= <Sunny, Warm, High, Strong, Cool, Same>

x_2= <Sunny, Warm, High, Light, Warm, Same>

h_1= <Sunny, ?, ?, Strong, ?, ?>

h_2= <Sunny, ?, ?, ?, ?, ?>

h_3= <Sunny, ?, ?, ?, Cool, ?>

FIND-S: Finding a Maximally Specific Hypothesis

First considering the most specific hypothesis and then generalizing* it each time it to be consistent over all positive examples

h_0 = <Φ, Φ, Φ, Φ, Φ, Φ>

h_1 = <*Sunny, Warm, Normal, Strong, Warm, Same*>

h_2 = <*Sunny, Warm,* ?, *Strong, Warm, Same*>

h_3 = <*Sunny, Warm,* ?, *Strong, Warm, Same*>

h_4 = <*Sunny, Warm,* ?, *Strong,* ?, ?>

Maximally specific hypothesis consistent with all positive examples

**Considering EnjoySport data set mentioned in the textbook*

FIND-S: Finding a Maximally Specific Hypothesis (Cont.)

The Algorithm Pseudocode:

---

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
    - For each attribute constraint a_i in h
        If the constraint a_i is satisfied by x
        Then do nothing
        Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Version Spaces & CANDIDATE-ELIMINATION Algorithm

- **Concern of FIND-S Output Hypothesis:** It is just one of many hypotheses that might fit training data equally well
- **Advantage of CANDIDATE-ELIMINATION Algorithm:** Outputs a description of the set of all hypotheses consistent of training examples
- A hypothesis h is consistent with a set of training examples D if and only if $h(x) = c(x)$ for each example $\langle x, c(x) \rangle$ in D
- $\text{Consistent}(x, D) = (\forall \langle x, c(x) \rangle \in D)\ h(x) = c(x)$
- Version Space is subset of all hypotheses with respect to hypothesis space H and the training data D, because it contains all plausible version of target concept
- $VS_{H,D} = \{h \in H \mid Consistent(h, D)\}$

LIST-THEN-ELIMINATE Algorithm

- LIST-THEN-ELIMINATE algorithm first initializes the version space with all hypotheses in *H*, and then eliminates any hypothesis found inconsistent with any training data
- It can output just one or more hypotheses consistent with observed data
- The Algorithm Pseudocode:

The LIST-THEN-ELIMINATE Algorithm

1. $VersionSpace \leftarrow$ a list containing every hypothesis in H
2. For each training example, $\langle x, c(x) \rangle$

   remove from $VersionSpace$ any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in $VersionSpace$

CANDIDATE-ELIMINATION Learning Algorithm

- Represents a version space storing only its most general members (*G*) and its most specific (*S*).
- Given these two sets *S* and *G*, it is possible to enumerate all members of the version space as needed by generating the hypotheses that lie between these two sets

S_0 <-- {<Φ, Φ, Φ, Φ, Φ, Φ>

G_0 <-- {<?, ?, ?, ?, ?, ?>}

- As each training example is considered, the *S* and *G* boundary sets are generalized and specialized, respectively, to eliminate any hypotheses found inconsistent with the new training example
- After processing of all examples, the computed version space contains all the hypotheses consistent with these examples and only these hypotheses

CANDIDATE-ELIMINATION Learning Algorithm (Cont.)

The Algorithm Pseudocode:

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

Remarks on VERSION SPACES & CANDIDATE-ELIMINATION

- Convergence
  - •Convergence will take place provided
    - There are no error in the training examples
    - There is some hypothesis in *H* that correctly describes the target concept
  - •Training Examples Containing Errors
    - Hypothesis will be removed if it is inconsistent with the training examples.
    - With additional data learner will eventually detect inconsistency by noticing that *S* and *G* boundary sets converge to an empty version space with no hypothesis consistent with observed data
- Using Partially Learned Concepts
  - •Partially learned concept may contain more than one hypothesis in version space
  - •In this case, every hypothesis will classify the new examples during inference and select the mostly voted class by all the hypotheses

Inductive Bias

PLACE HOLDER

End-to-End Machine Learning Project

Major Steps Involved

1. Looking at the big picture.
2. Getting the data.
3. Discovering and visualizing the data to gain insights.
4. Preparing the data for Machine Learning algorithms.
5. Selecting a model and train it.
6. Fine-tuning your model.
7. Presenting your solution.
8. Launching, monitoring, and maintaining your system.

The Big Picture

- **The Context:** Automatic prediction of district's median house prices using census data
- **Framing the Problem:**
  - Understanding current business problem
    - Time consuming and costly manual operations
    - Performance is not always good
  - Proposed solution
    - A model to predict house price
      - Considering if it should be a supervised, unsupervised or reinforcement learning
      - Considering if it should be classification, (univariate or multivariate) regression or any other type of task
      - Whether it should be batch or online learning
    - Downstream apps to consume the predictions for further business processes and to decide on investments

The Big Picture (Cont.)

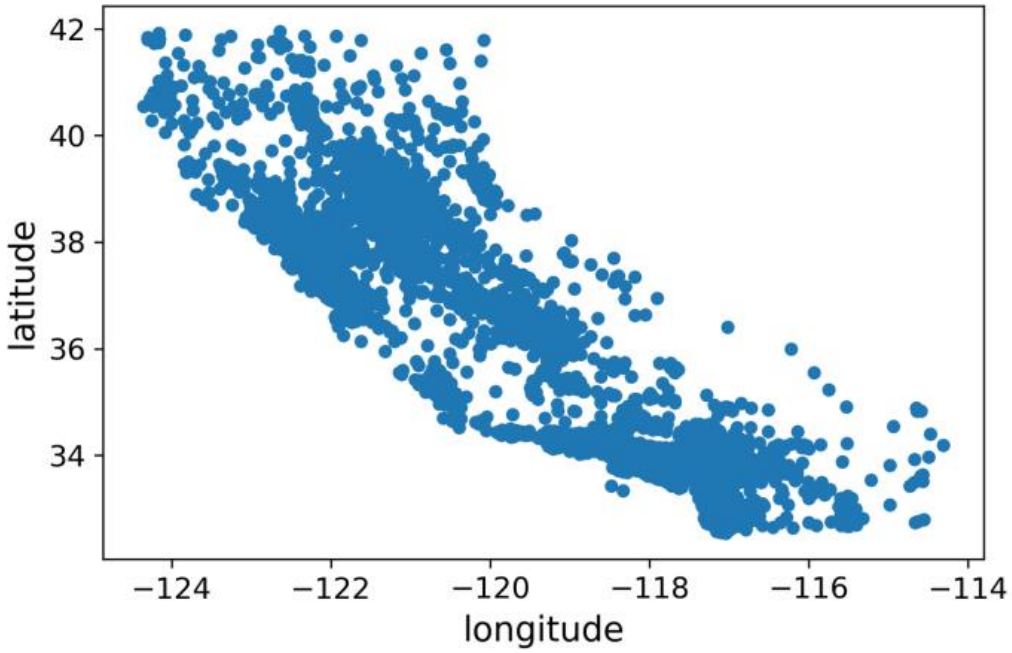
- **Deciding on performance metrics:**
  - Means Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
  - Mean Absolute Error (MAE)
- Choosing candidate algorithms
- Realizing development effort

Getting the Data

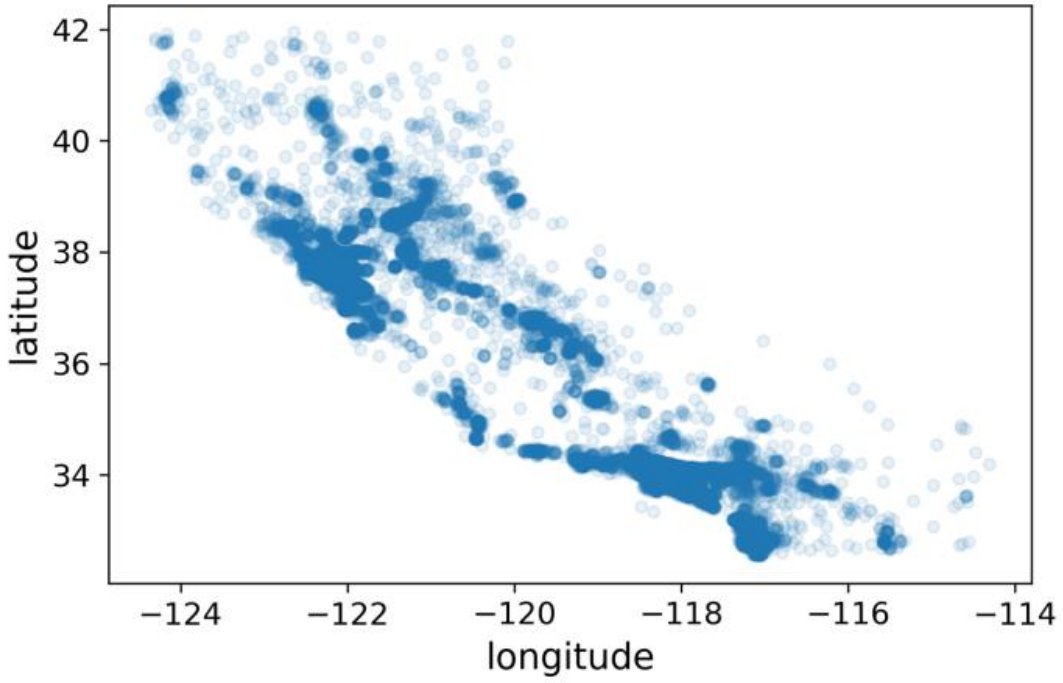
- Prerequisites (setting up development environment)
- Downloading the relevant data
- Taking a quick look at the data
  - Basic information about dataset
  - Elementary statistics
  - Visualizing
  - Distribution
- Creating test set
  - Randomizing the index of the observations
  - Splitting data into train and test data set (~80:20)
  - Considering stratification, if required

Exploring & Visualizing the Data

Visualizing geographic data



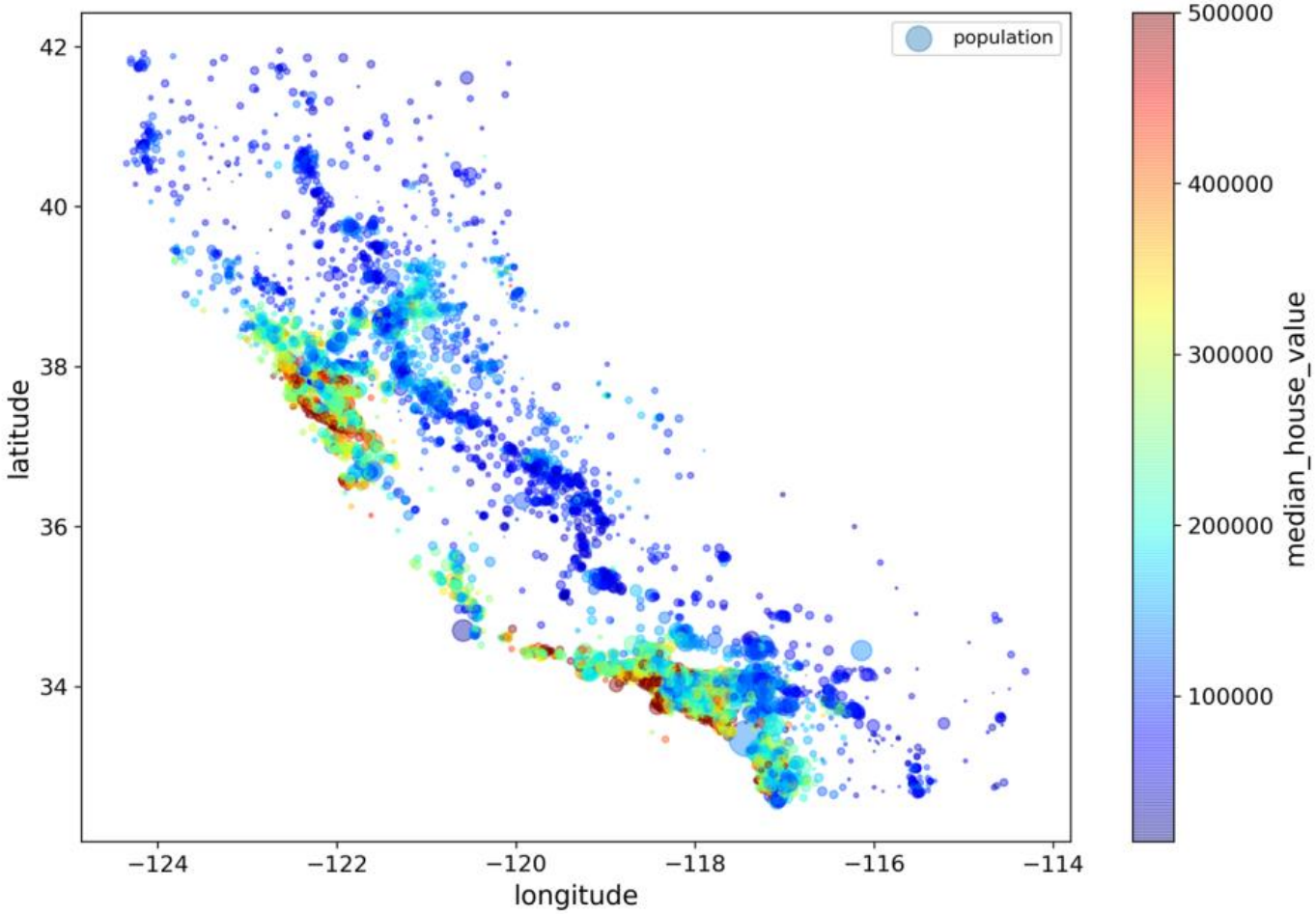
A geographical scatterplot



A better visualization of geographical data highlighting high-density areas

Exploring & Visualizing the Data (Cont.)

Better visualization

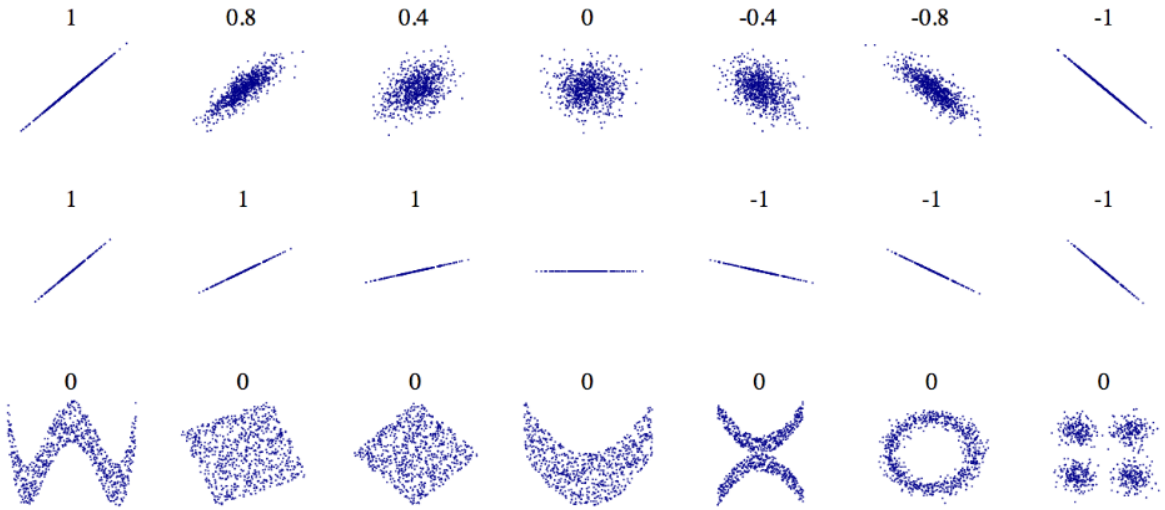


California housing prices: red is expensive, blue is cheap, larger circles indicate areas with a larger population

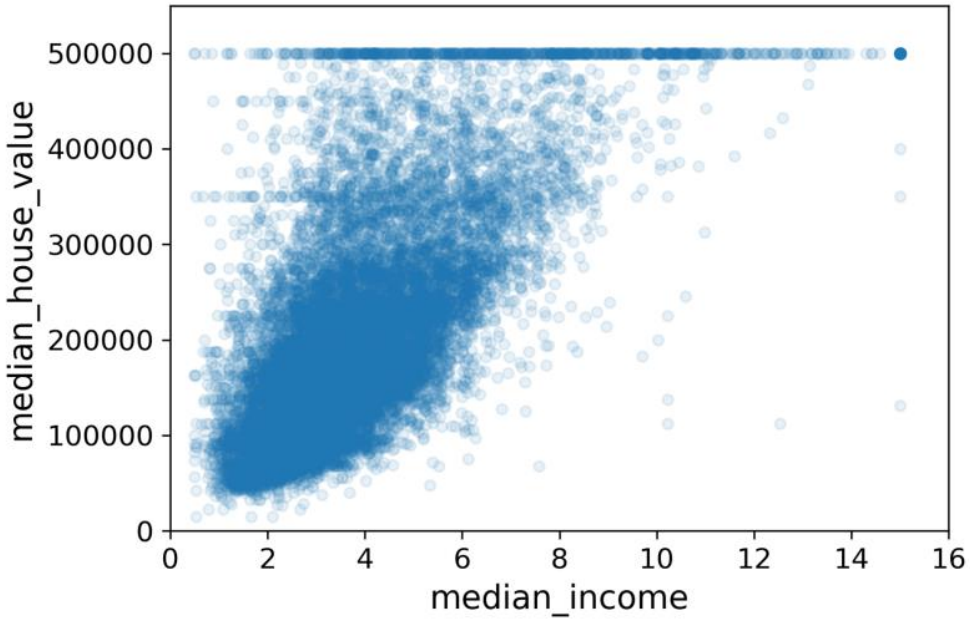
Exploring & Visualizing the Data (Cont.)

Finding correlations

- Correlation coefficient and its range
- Linear correlation: Positive & Negative correlation
- Nonlinear correlation



Standard correlation coefficient of various datasets



Strong relation between median income and medium house value

Preparing Data

- Data cleaning
  - Handling missing values
    - Removing relevant data, or
    - Removing attribute from entire dataset, or
    - Setting the values with some value (zero, mean, median, etc.)
  - Handling text and categorical attributes
    - Ordinal encoding
    - One-hot encoding
  - Transforming data
    - Scaling features
      - Min-max
      - Standardization
    - Pipelining for transformations

Selecting & Training a Model

- Training model
  - Simple models
  - Complex models
- Evaluating model
  - Evaluation on training set
  - Evaluation over cross-validation
- Considering better model (if applicable)

Fine-tuning Model

- Grid searching
- Randomized searching
- Ensembling
- Analyzing model performances
- Evaluating final model against test set

Launching, Monitoring & Maintaining Models

- Deployment
  - Deploying model as web service
  - Deploying model over Cloud

Model deployed as a web service and being consumed by an application

- Monitoring
  - Monitoring model performance on regular basis
  - Setting up alerts to be triggered when performance falls below threshold
- Maintenance
  - Scripting for automatic model building and hyperparameters tuning
  - Keeping model backup
  - Versioning data sets

Classification

Predicting Classes

MNIST – The *Hello World* of Machine Learning

- MINST – a data set consisting 70,000 small images of digits handwritten by school students and employees
- The shape of each image is 28 by 28 pixels and has been flattened as 784 features in the data set
- The range of each feature is 0 (white) – 255 (black)

Major Steps Involved

- Types of Classification
  - Binary classifiers
  - Multiclass classifiers
  - Multilabel classifier

- Measuring performance
  - Accuracy using cross-validation
  - Precision, Recall, Confusion Matrix, F1 Score
  - Precision/Recall Trade-of
  - Receiver Operating Characteristics (ROC) Curve

Performance Metrics

- Accuracy
- Precision, Recall & F1 Score
- ROC Curve
- ROC AUC
- False Positive Rate (Recall or Sensitivity)
- True Positive Rate
- True Negative Rate (Specificity)

$$Accuracy = \frac{Total\ number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

$$= \frac{TP+TN}{TP+TN+FP+FN}$$

Multiclass/Multinomial Classifiers

- **Models:** Logistic Regression, Support Vector Machine (SVM) Classifier
- **Designing Multiclass Classifier from Multiple Binary Classifiers**
  - **One-versus-the-Rest (OvR)** or **One-versus-All (OvA)** Strategy
    - Training each binary classifier for one class
    - Prediction (decision score) is received from all classifiers
    - Selecting class from the classifier with highest decision score
  - **One-versus-One (OvO) Strategy**
    - Training each binary classifier for each pair of class
    - For ***N*** classes, ***N * (N – 1) / 2*** number of binary classifiers will be required
    - Prediction (decision score) is received from all classifiers
    - Selecting the class from the classifier with highest decision score
    - Each classifier needs to be train only on the part of the data set for the two classes it must predict
  - OvR is preferred for most binary classifiers except few such as SVM classifier that scales poorly with the size of the training set and for these algorithms OvO is preferred due to smaller data sets.

Entropy

- It's a measure commonly used to in Information Theory that characterizes the impurity or purity of an arbitrary collection of observations.

$$Entropy(S) = \sum_{i=1}^{n} -p_i * log_2 p_i$$

Where p_i is proportion of *S* belonging to class *i*

Information Gain

- It's a statistical property that measures how well a given an attribute separates observations according to their target classification.
- Algorithms such as *Iterative Dichotomizer 3* (ID3) uses this information gain measure to select among candidate attributes at each step while growing the decision tree.

$$Information\_Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} * Entropy(S_v)$$

where,

- *Values(A)* is set of all possible values in attribute *A*
- S_v is the subset of *S* for which attribute *A* has value *v* (i.e. $S_v = \{s \in S | A(s) = v\}$)

Training Models

Knowing how things work: Right algorithms, good set of hyperparameters, debugging and performing error analysis

Linear Regression

- Different ways to train using:
  - Direct "closed-form" equation to compute model parameters that minimizes cost function over training set
  - Iterative optimization called Gradient Descent (GD) to gradually tweaking model parameters towards minimizing cost function over training set

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$\hat{Y}$: Predicted value

n: Number of features

x_i: i^{th} feature value

θ_i: i^{th} model parameter (w_0 is bias or intercept term, and feature weights θ_1, θ_2, …, θ_n)

Linear Regression...Cont.

- Vectorized Form:

$$\hat{y} = h_\theta(x) = \theta . x$$

θ is the model's parameter vector, containing the bias term θ_0 and the feature weights θ_1 to θ_n

x is the instance's feature vector, containing x_0 to x_n, with x_0 always equal to 1

$\theta \cdot x$ is the dot product of the vectors θ and x, which is equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$ and

h_θ is the hypothesis function, using the model parameters θ

Linear Regression...Cont.

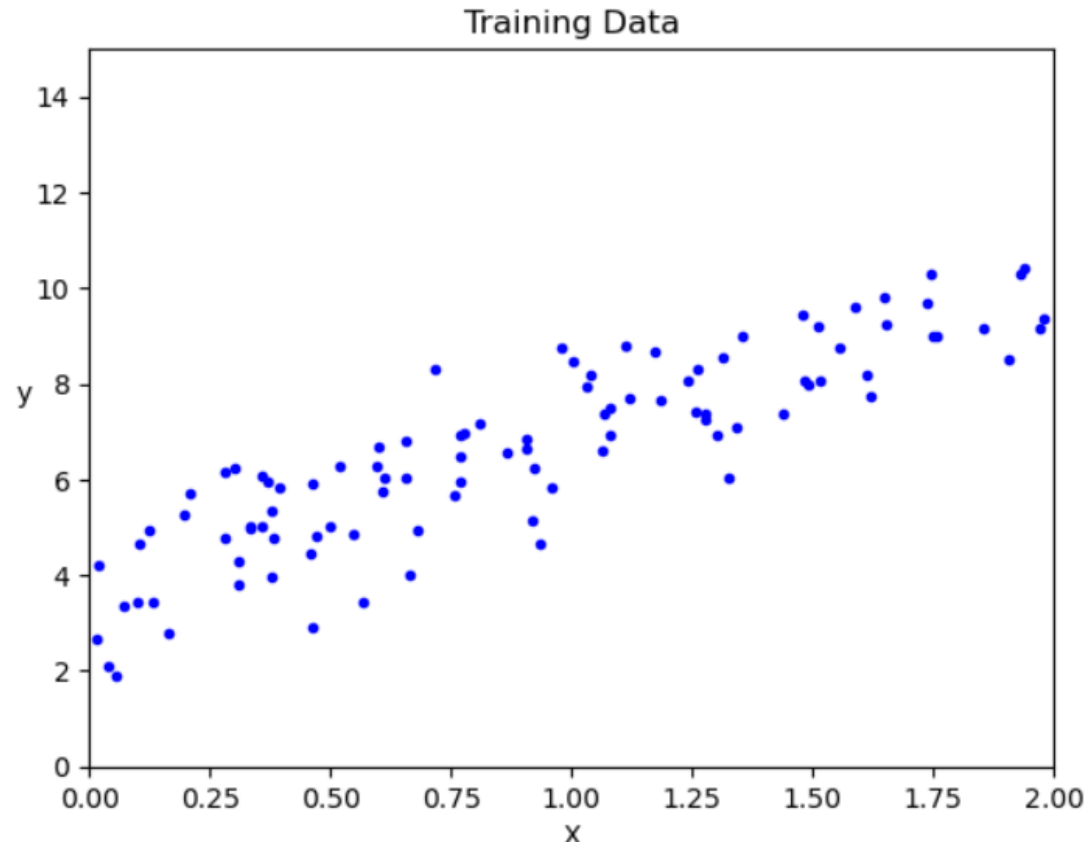
- Normal Equation or Direct "closed-form" Equation to Compute Model Parameters

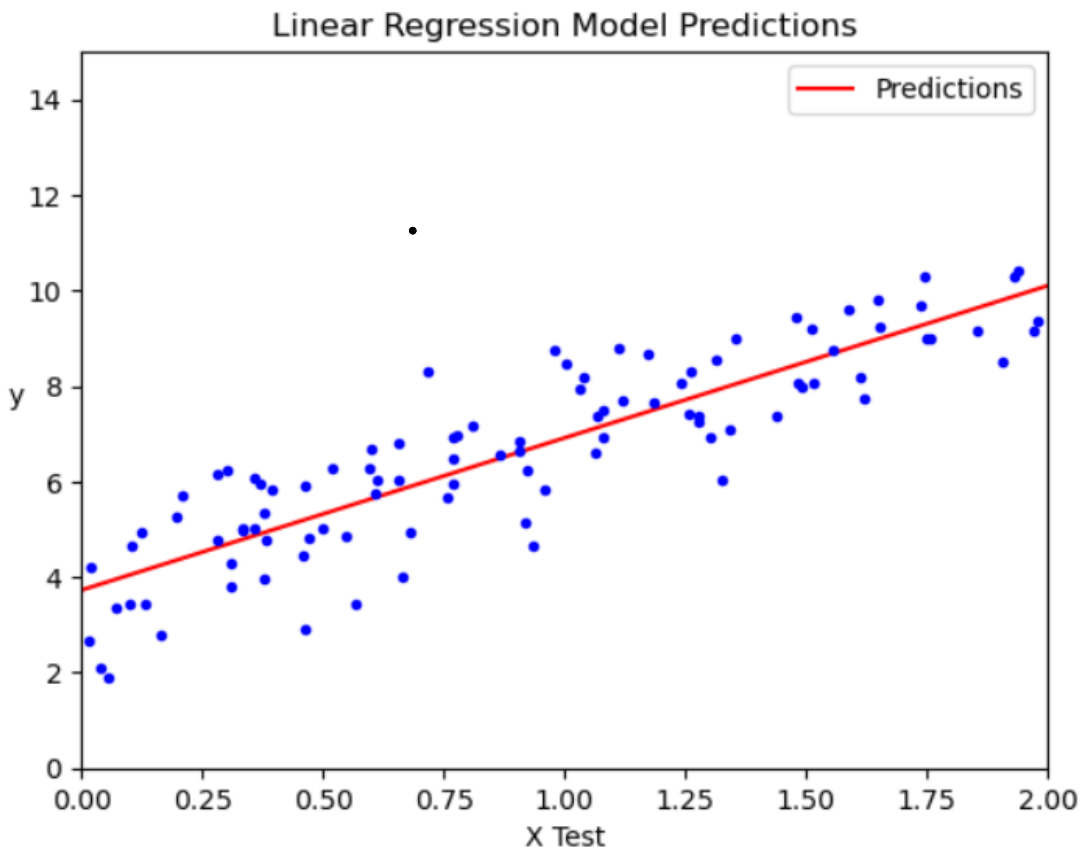
$$\hat{\theta} = (X^T X)^{-1} X^T y$$

θ: Value of θ that minimizes the cost function,

y: vector of target values containing $y^{(1)}$ to $y^{(m)}$

Linear Regression...Cont.





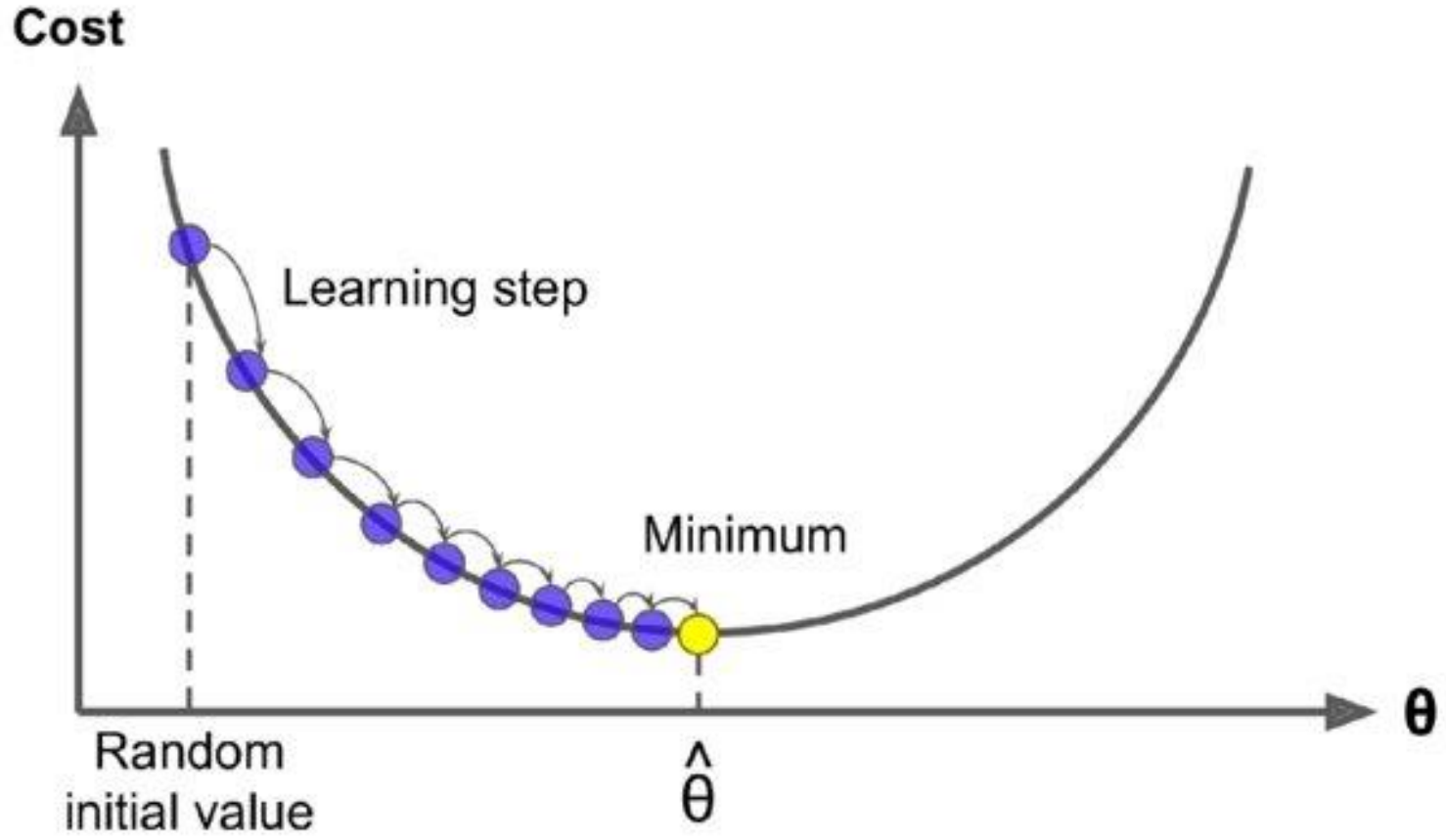
Linear Regression...Cont.

- Computational Complexity for Normal Equation
  - Computes inverse of $X^T X$, which is an $(n + 1) \times (n + 1)$ matrix (for n features)
  - Computational complexity of inverting such a matrix is ~ $O(n^{2.4})$ to $O(n^3)$
  - For example, doubling the number of features, the computation time gets multiplied roughly by $2^{2.4} = 5.3$ to $2^3 = 8$
- Computational Complexity for SVD based approach (e.g. `numpy.linalg.lstsq()`)
  - It is about $O(n^2)$
  - For example, doubling the number of features, the computation time gets multiplied roughly by $2^2 = 4$

Gradient Descent

- A generic optimization algorithm that produces parameters to minimize a cost function
- It measures the local gradient of the error function with respect to parameter vector θ and goes in the direction of descending gradient till the gradient becomes zero.
- Steps:
  - θ is initialized with random values
  - Iterates gradually in steps to improve on θ attempting to minimize a cost function (e.g. MSE) until it converges to a minimum (refer figure)

Gradient Descent...Cont.

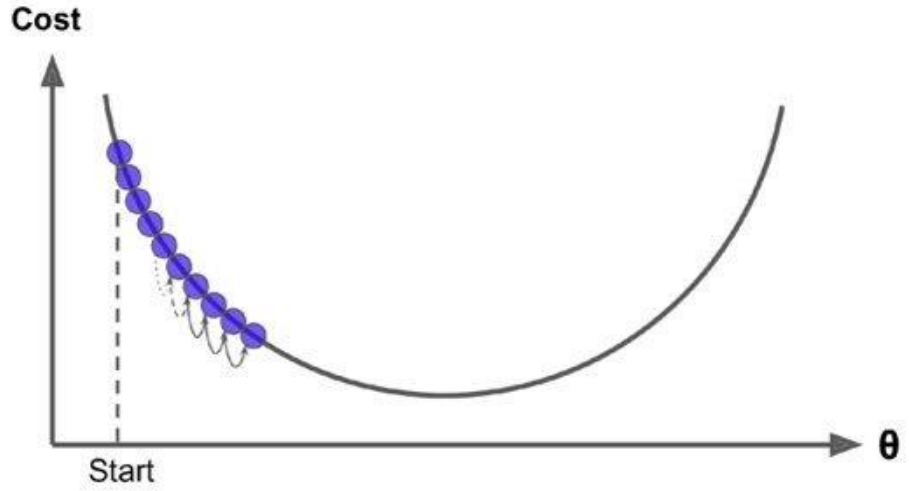


Learning step size is proportional to slope of cost function

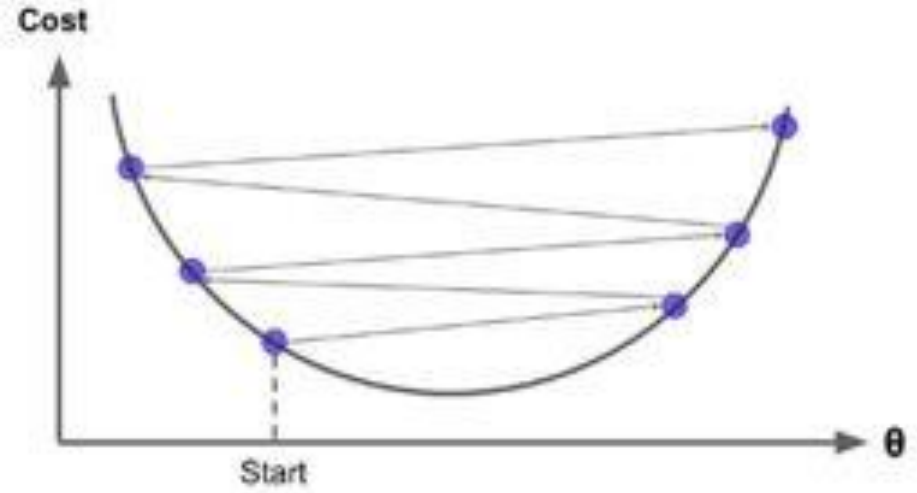
Gradient Descent...Cont.

Learning Rate

- Size of the step is determined by learning rate
- If Learning Rate is small, then algorithm will have to go through many iterations to converge consuming longer time
- If Learning Rate is too high, then algorithm may jump across other side of the gradient possibly even higher up than it was before resulting divergence



Low learning rate

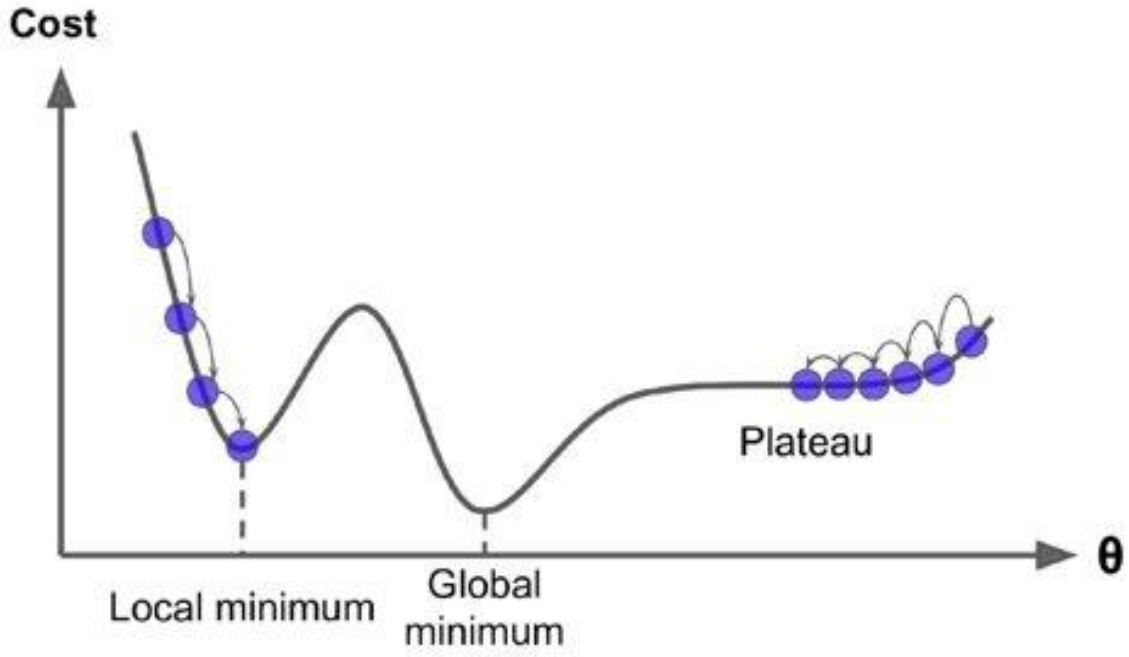


High learning rate

Gradient Descent...Cont.

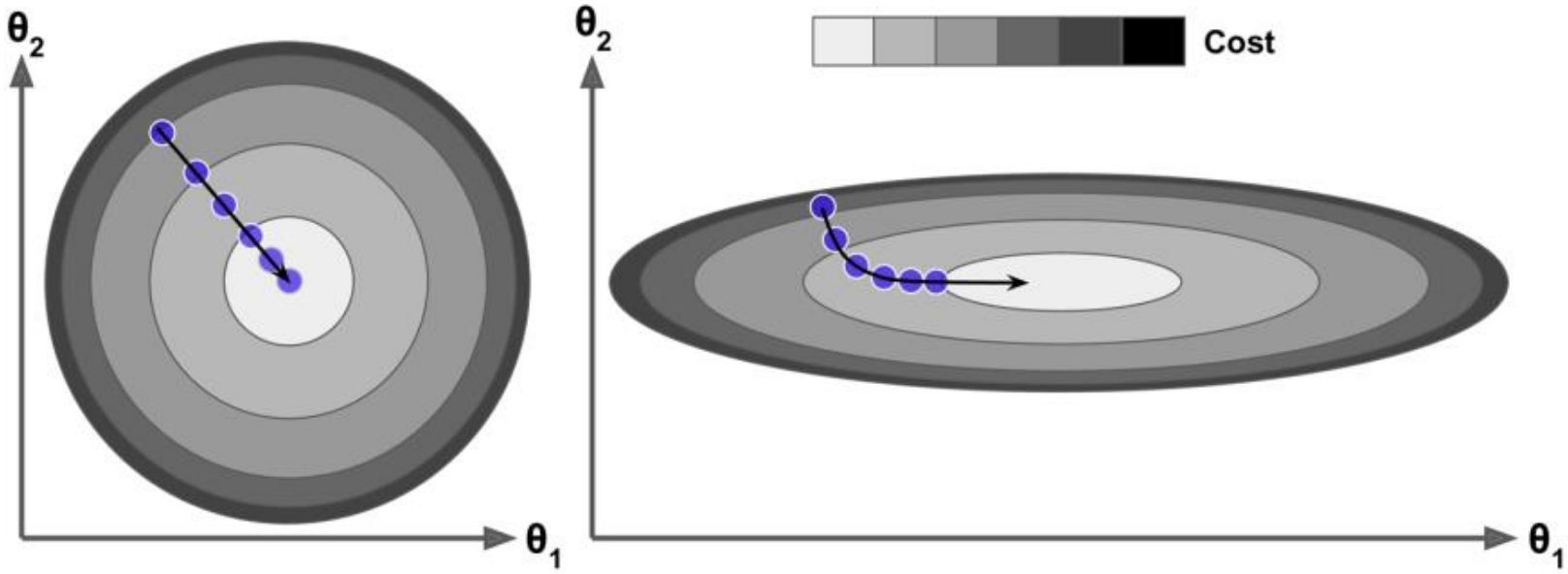
Local & Global Minimum

- If the random initialization starts the algorithm on the left, then it will converge to a local minimum
- If it starts on the right, then it will take a very long time to cross the plateau
- Stopping too early, it will never reach the global minimum
- Fortunately, MSE cost function for Linear Regression model is a convex function meaning there are no local minimum



Local & Global Minimum

Gradient Descent…Cont.



Features having same scale

Features having different scales

- Cost function can be elongated if features have very different scales
- Features should be scaled for quicker convergence
- Search space for model parameters increases with the increase in number of parameters and harder it becomes to converge

Batch Gradient Descent (BGD)

- **Partial Derivative:** Gradient of cost function is computed with respect to each model parameter θ_j and denoted by

$$\frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\top} \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Batch Gradient Descent (BGD)...Cont.

- Instead of computing these partial derivatives individually, these are computed all in one go
- This computation involves calculations over full training set at each Gradient Descent step
- This makes it slow especially for large training data set
- But as compared with Normal Equation or SVD, it scales well with number of features

$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^{\mathsf{T}} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

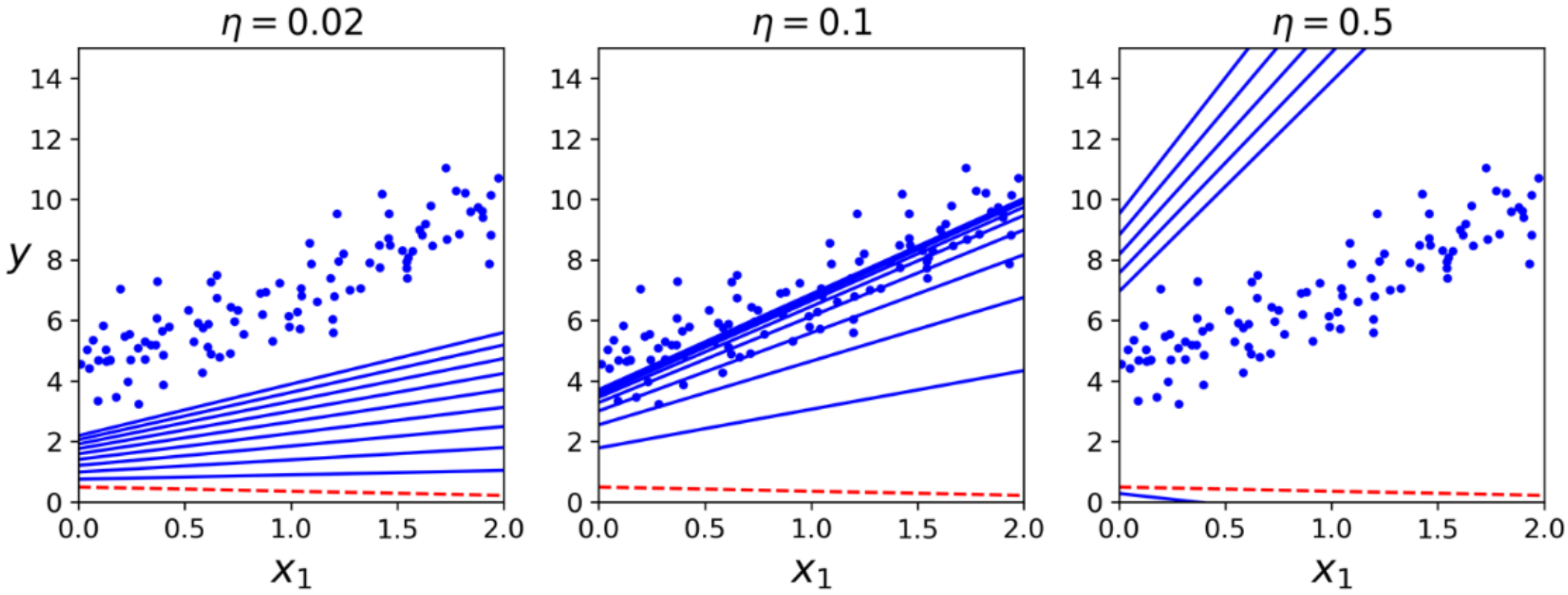
Gradient vector of cost function

Batch Gradient Descent (BGD)...Cont.

- Gradient vector $\nabla_\theta MSE(\theta)$ is multiplied by η (eta) to determine downhill step size.

$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta MSE(\theta)$$

Batch Gradient Descent (BGD)...Cont.



Gradient Descent with various learning rates

Batch Gradient Descent (BGD)...Cont.

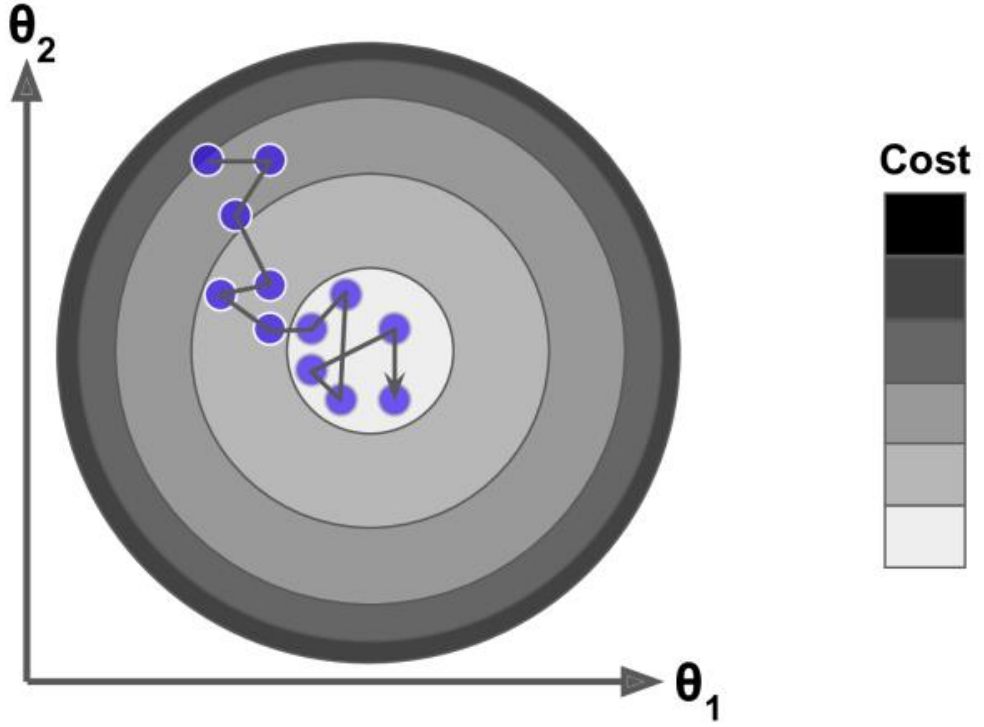
- A good learning rate can be found out over grid search
- Maximum iterations can be set for grid search to eliminate models that take too long to converge
- If number of iterations is low, the solution could be suboptimal if the algorithm stops.
- If number of iterations is high, there could be wastage of time as the model parameters do not change anymore
- Simply solution would be to set very large number of iterations, but to interrupt algorithm when gradient vector becomes tiny i.e. when its norm becomes smaller than a tiny number ε (called the tolerance)
- **Convergence Rate:** Gradient Descent can take $O(1/\varepsilon)$ iterations for convergence depending on the shape of the cost function

Stochastic Gradient Descent (SGD)

- Batch Gradient Descent uses whole training set to compute gradients at every step making it slow especially for larger training set
- SGD picks one instance from training set randomly at every step to compute the gradients on that instance
- It is faster as it manipulates little data at every iteration
- It also makes it possible train on huge training set since only one instance needs to be in memory at each iteration (an out-of-core algorithm)
- It is irregular than BGD. The cost function bounces up and down decreasing on average. It closes to the minimum with good parameters value, but not optimal.

Stochastic Gradient Descent (SGD)…Cont.

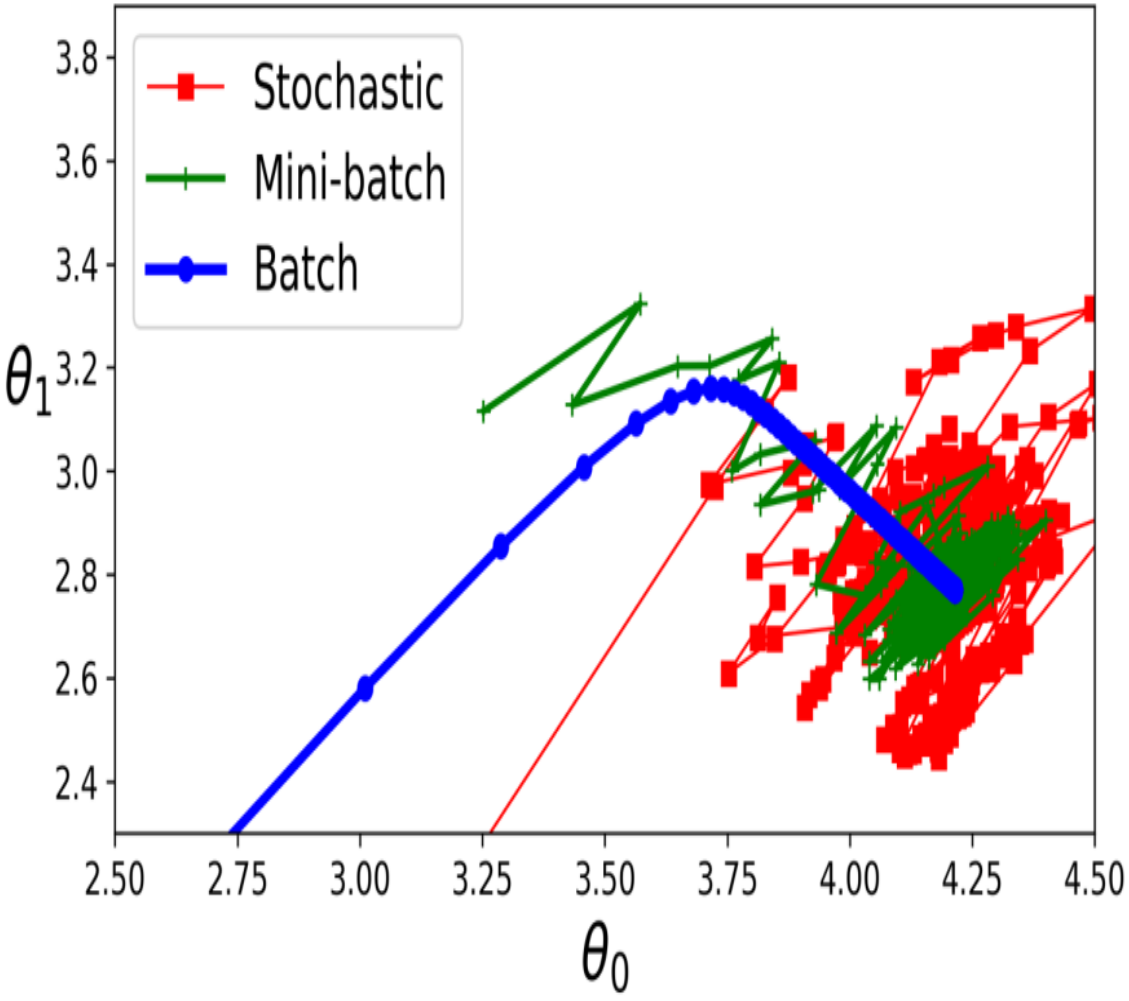
- Due to randomness, SGD has better chance to find global minimum
- But it also means, it doesn't settle at minimum
- One solution is to start with larger learning rate and reduce it gradually
- Learning rate schedule function determines learning rate at each iteration
- If the learning rate is reduced too quickly, it will get stuck in local minimum
- If the learning rate is reduced too slowly, it will jump around minimum for long time



Stochastic Gradient Descent

Mini-batch Gradient Descent

- It computes gradients on small random sets of instances callled mini-batches
- Getting performance boost from hardware optimization of matrix operation especially when using GPUs
- It ends up walking around a bit closure to the minimum than SGD, but it may be harder to escape local minimum
- BGD stops at minimum while Mini-batch and SGD continue to walk around, but former takes lot of time in each step
- A good learning schedule helps Mini-batch and SGD to reach minimum



Gradient Descent paths in parameter space

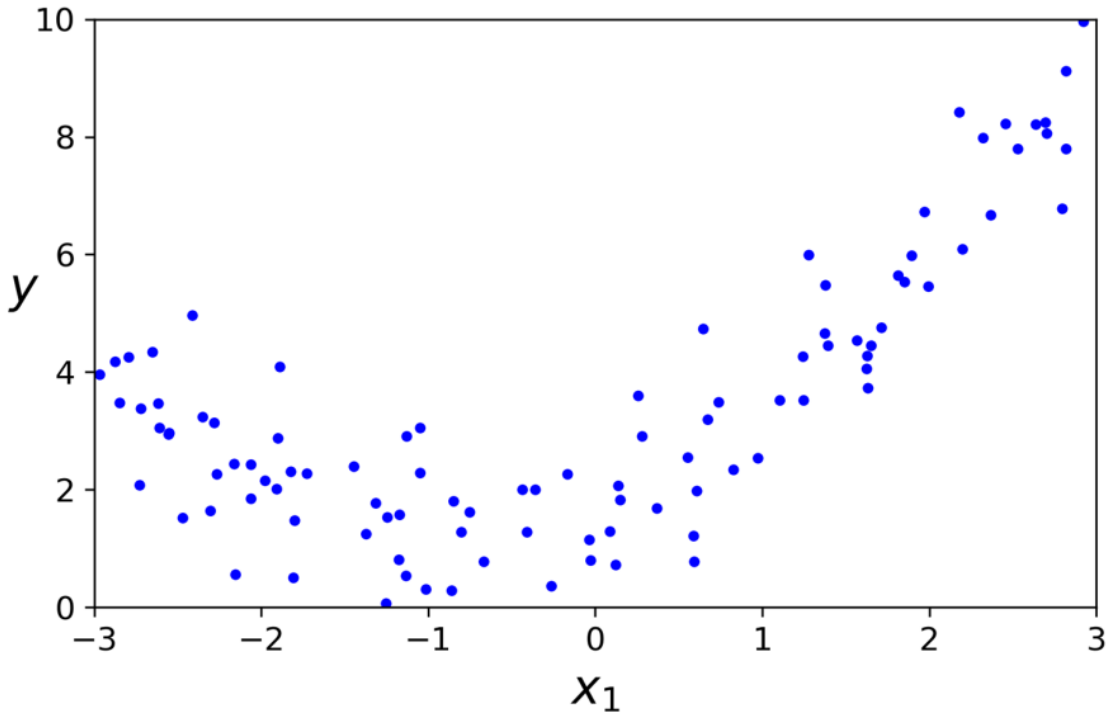
Comparison of Linear Regression Algorithms

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

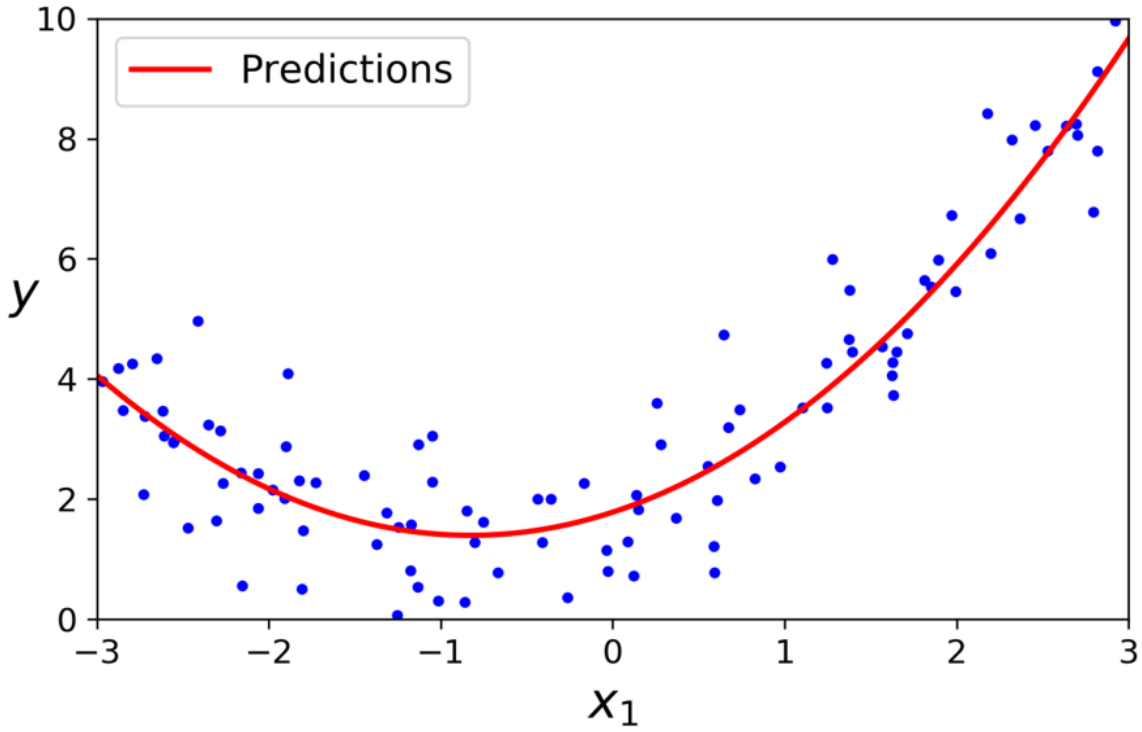
m is the number of training instances
n is the number of features

Polynomial Regression

- It fits non-linear data
- Adding power to each feature as new feature and then training a linear model to these set of features would be a simple solution



Non-linear and noisy dataset



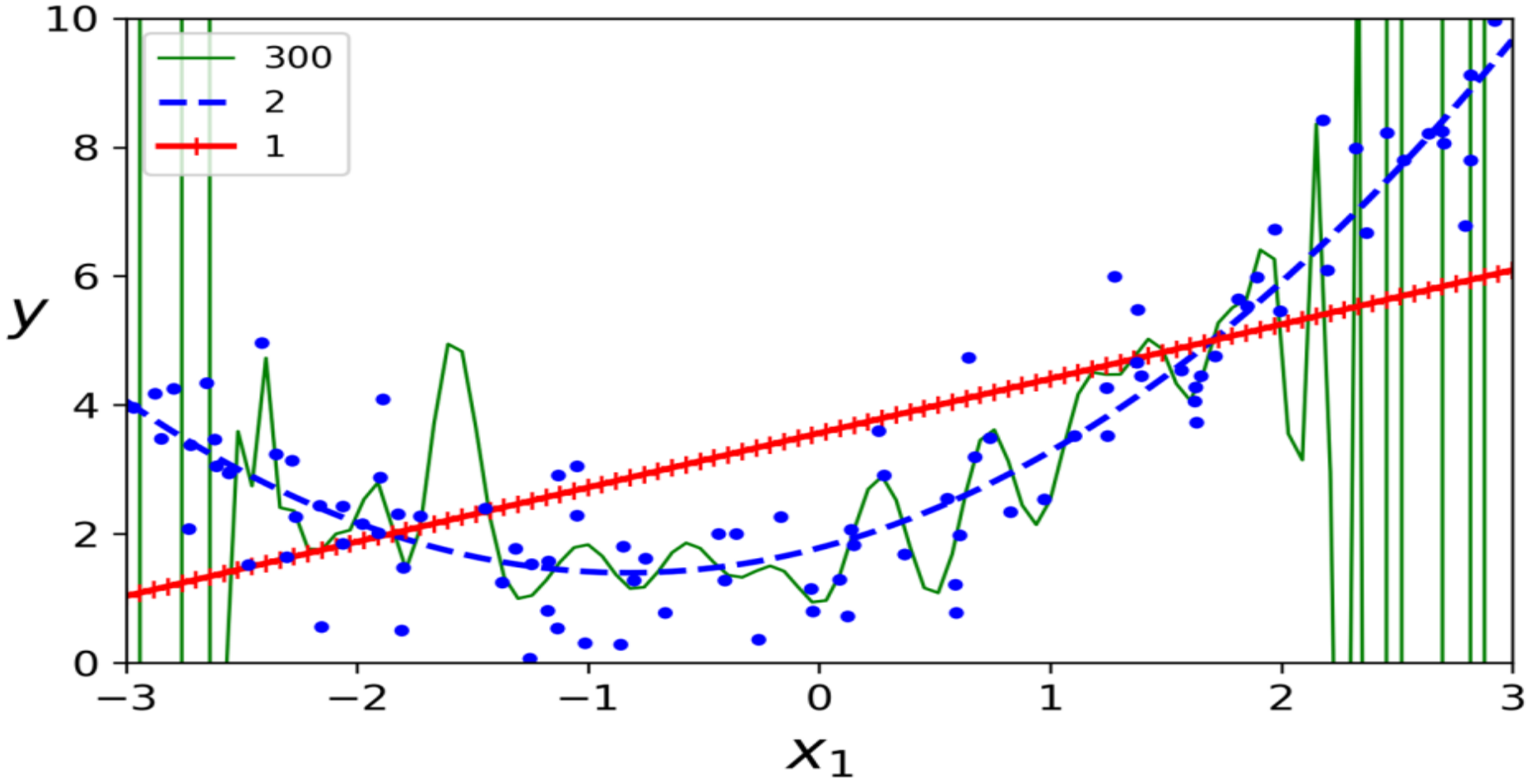
Prediction using Polynomial Regression

Polynomial Regression...Cont.

- For multiple features, Polynomial Regression can find relationship between features
- For example, if *a* and *b* are the two features, then Polynomial Regression with degree = 3 would add features a^2, a^3, b^2, b^3, ab, a^2b and ab^2
- In general, PolynomialFeatures(degree=d) transforms an array containing *n* features into an array containing $(n + d)! / d!n!$ features, where $n!$ is the factorial of *n*

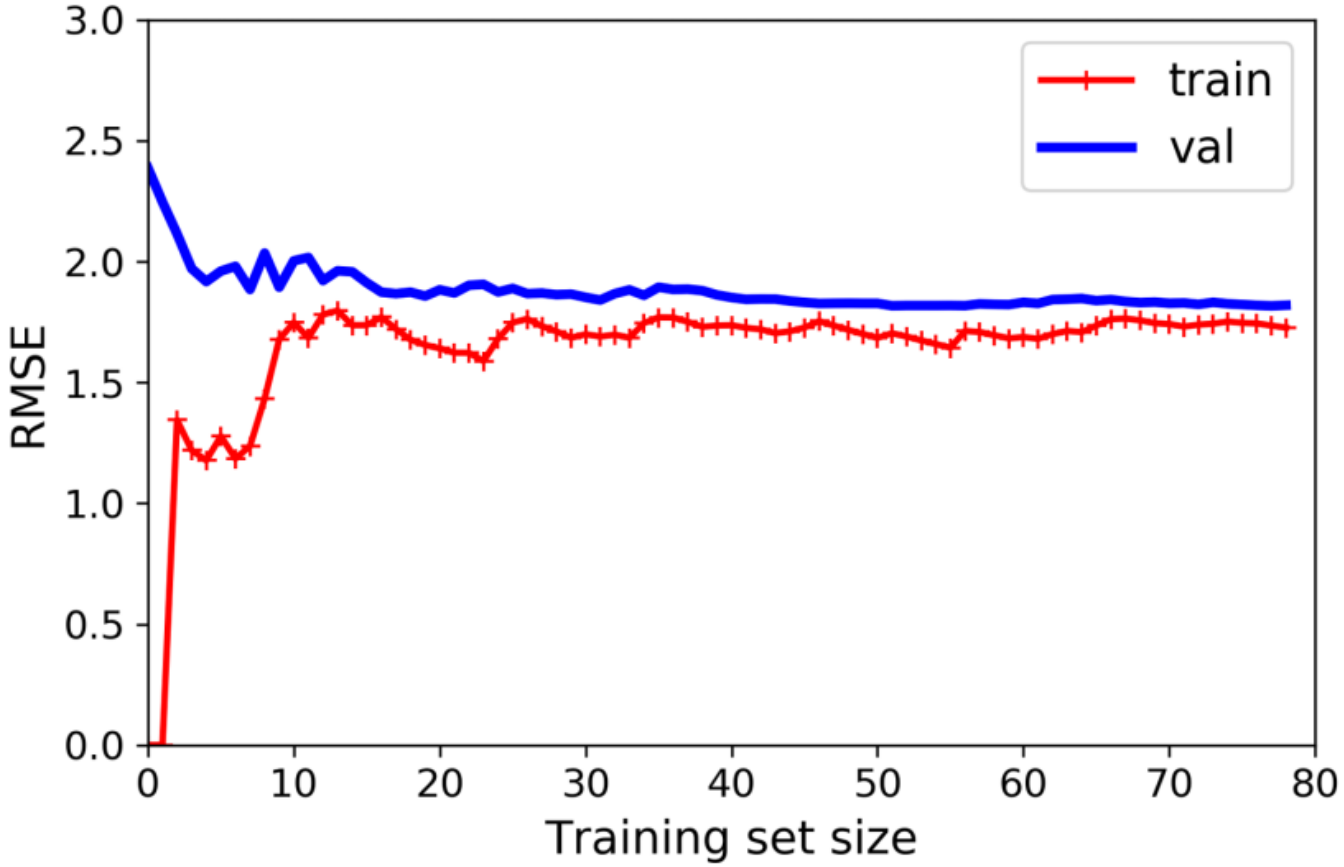
Learning Curves

- High-degree Polynomial Regression may overfit training data



Learning Curves…Cont.

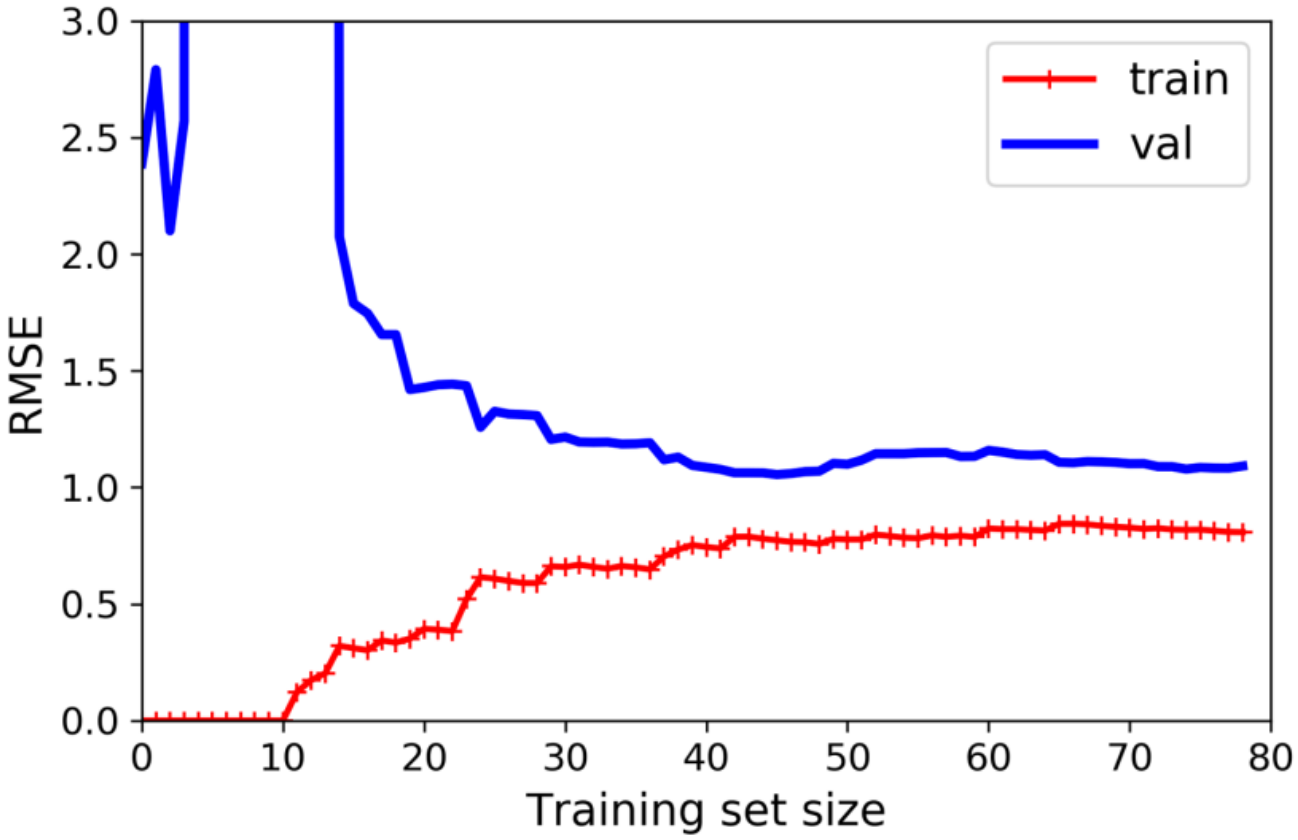
- Learning Curves are the plots of model's performance on training and validation set as a function of training set size or training iteration
- To generate the plots, model is trained several times on different sized subsets of the training set



Learning curves for Linear Regression model

Learning Curves…Cont.

- The error out of 10-degree Polynomial Regression model on training data is much lower than with the Linear Regression model
- Gap between the curve indicates overfitting as model performs better on training set than on validation set
- Larger training set would bring these curves closure



Learning curves for 10-degree Polynomial Regression model

Learning Curves…Cont.

Bias/Variance Trade-off

Bias: It is a generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A highly biased models are prone towards underfitting

Variance: This part is due to the model's excessive sensitivity to small variations in the training data. A model with higher degrees of freedom is likely to have high variance and thus overfits the training data

- Increasing model's complexity will increase its variance and reduce its bias
- Reducing model's complexity will increase its bias and reduce its variance

Regularized Linear Models

- Ways to reduce overfitting
  - **Regularizing:** A simple way to regularize a polynomial model is to reduce number of polynomial degrees
  - **Constraining:** For a linear model, regularization is typically achieved by constraining the weights of the model. Ridge, Lasso and Elastic Net are examples of regularized linear regression models

Regularized Linear Models…Cont.

Ridge Regression

- Regularization term $\alpha \sum_{i=1}^{n} \theta_i^2$ is added to cost function
- This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible
- Regularization term is added to the cost function only during training. Unregularized performance measure is used to evaluate the model's performance

α: Hyperparameter to control how much to regulate

n: Numbe of features

θ_i: Weight of i^{th} feature. Bias term θ_0 is not regularized

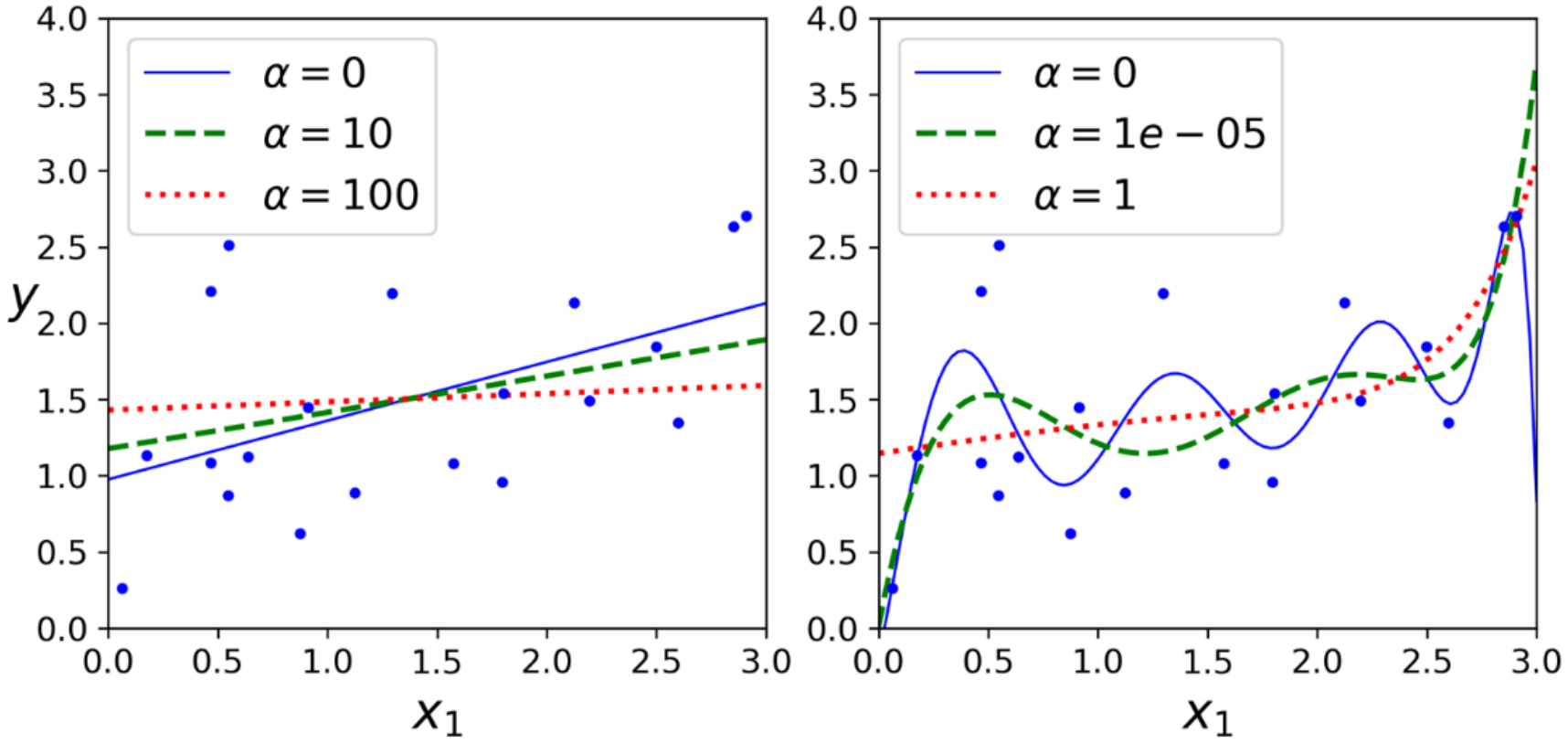
Regularized Linear Models…Cont.

Ridge Regression...Cont.

- If **α = 0**, then Ridge Regression is just Linear Regression. If **α** is very large, then all weights end up very close to zero resulting a flat line going through the data's mean
- Redge Regression Cost Function: $J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha\frac{1}{2}\Sigma_{i=1}^{n}\theta_i^2$
- **$J(\theta) = MSE(\theta) + \alpha.\frac{1}{2}.(||w||_2)^2$** where
  - w is vector of feature weights,
  - **$\frac{1}{2}.(||w||_2)^2$** is the regularization term
  - **$||w||_2$** is the **l_2** norm weight vector
- Data should be scale as Ridge Regression is sensitive to the scale of input features

Regularized Linear Models...Cont.

Ridge Regression...Cont.



A linear model (left) and a polynomial model (right), both with various levels of Ridge regularization

Regularized Linear Models…Cont.

Ridge Regression...Cont.

- Performing Ridge Regression by
  - computing a closed-form equation $\boldsymbol{\theta = (X^T X + \alpha A)^{-1} X^T y}$ or by
  - performing Gradient Descent by passing $\boldsymbol{l_2}$ as type of regularization term

A is (n+1) x (n+1) identity matrix except with a zero (0) in top-left cell

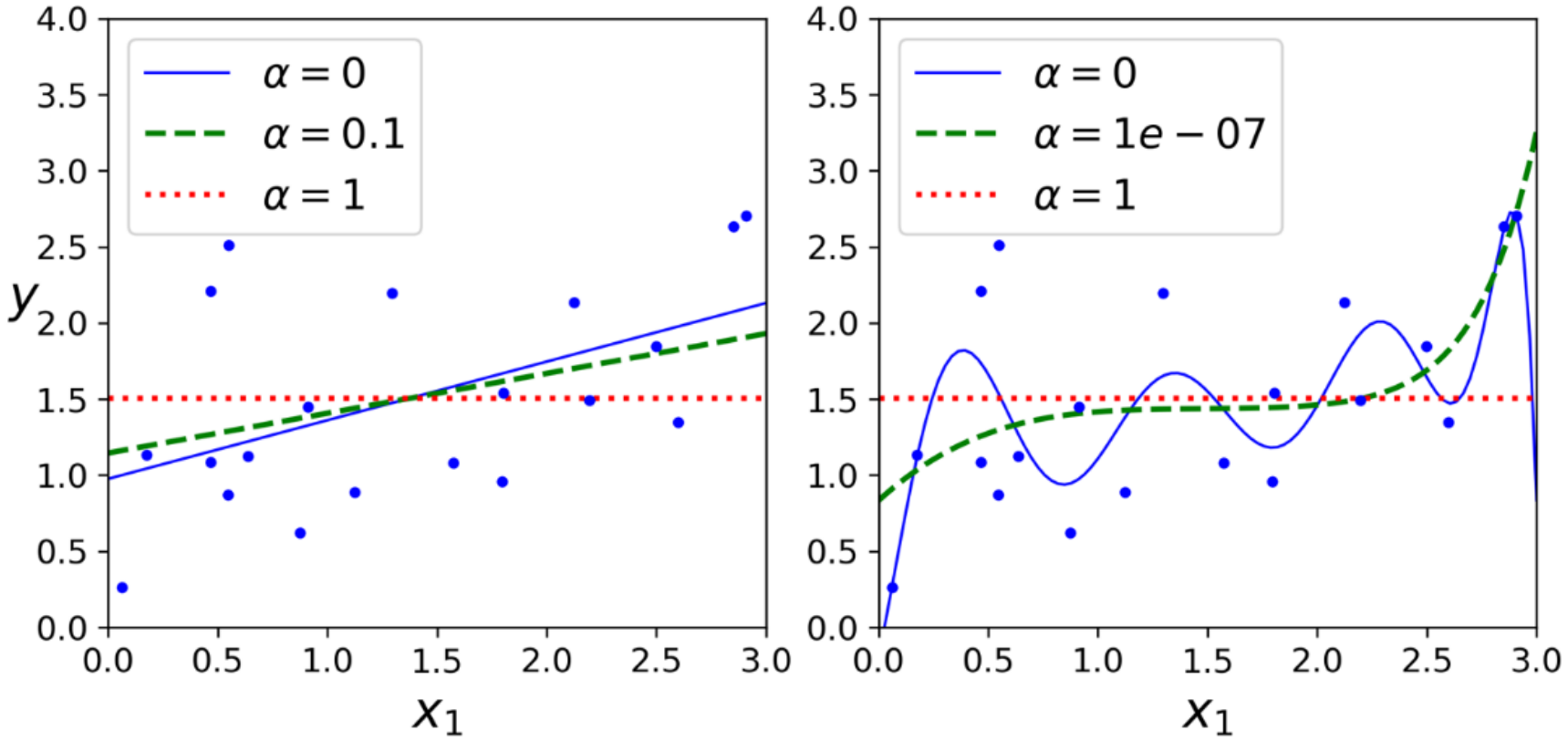
Regularized Linear Models…Cont.

Least Absolute Shrinkage and Selection Operator (Lasso) Regression

- It is like Ridge Regression, but adds it uses $\boldsymbol{\iota}_1$ norm of the weight vector instead of $\boldsymbol{\iota}_2$ norm
- Lasso Regression Cost Function: $J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} \left|\theta_i\right|$

Regularized Linear Models...Cont.

Lasso Regression...Cont.



A linear model (left) and a polynomial model (right), both using various levels of Lasso regularization

Regularized Linear Models…Cont.

Elastic Net

- It's middle ground between Ridge Regression and Lasso Regression
- The regularization term is a simple mix of both Ridge and Lasso's regularization terms, can be controlled the mix ratio r
- When r = 0, Elastic Net is equivalent to Ridge Regression, and when r = 1, it is equivalent to Lasso Regression
- Elastic Net Cost Function: $J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\alpha\Sigma_{i=1}^{n}\left|\theta_i\right| + \frac{1-r}{2}\alpha\Sigma_{i=1}^{n}\theta_i^2$

Regularized Linear Models…Cont.

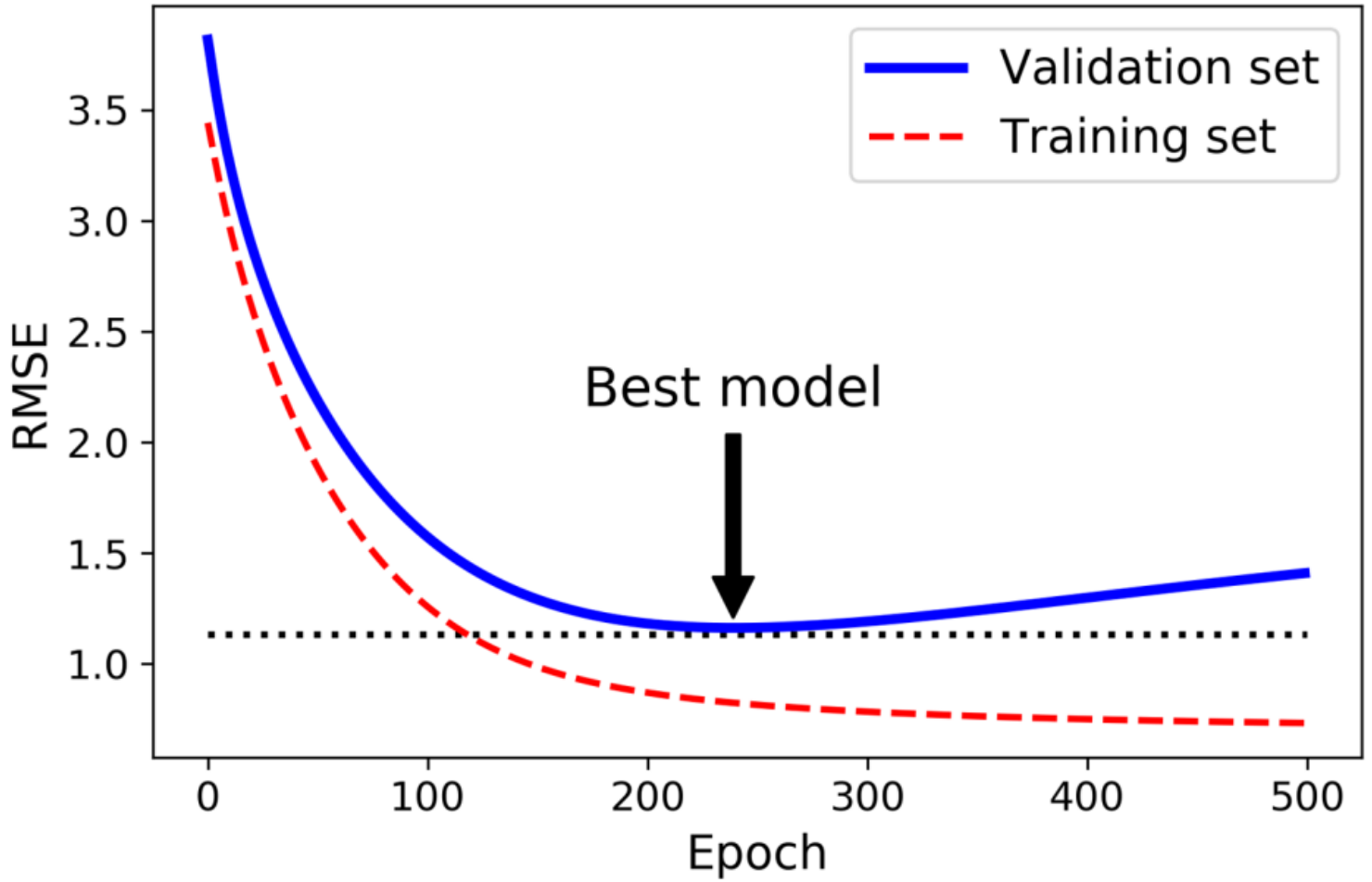
General Rules of Applying Regularization

- Plain Linear Regression should be avoided as it is preferrable to have little regularization
- Lasso or Elastic Net should be preferable if few features are useful as these tend to reduce the useless features' weights down to zero
- Elastic Net is preferred over Lasso because Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated

Regularized Linear Models…Cont.

Early Stopping

Another way for Gradient Descent to stop training as soon as validation error reaches to a minimum.



Early stopping regularization in Batch Gradient Descent on a high-degree Polynomial Regression model

Epoch: It is the number of passes a training dataset takes around an algorithm

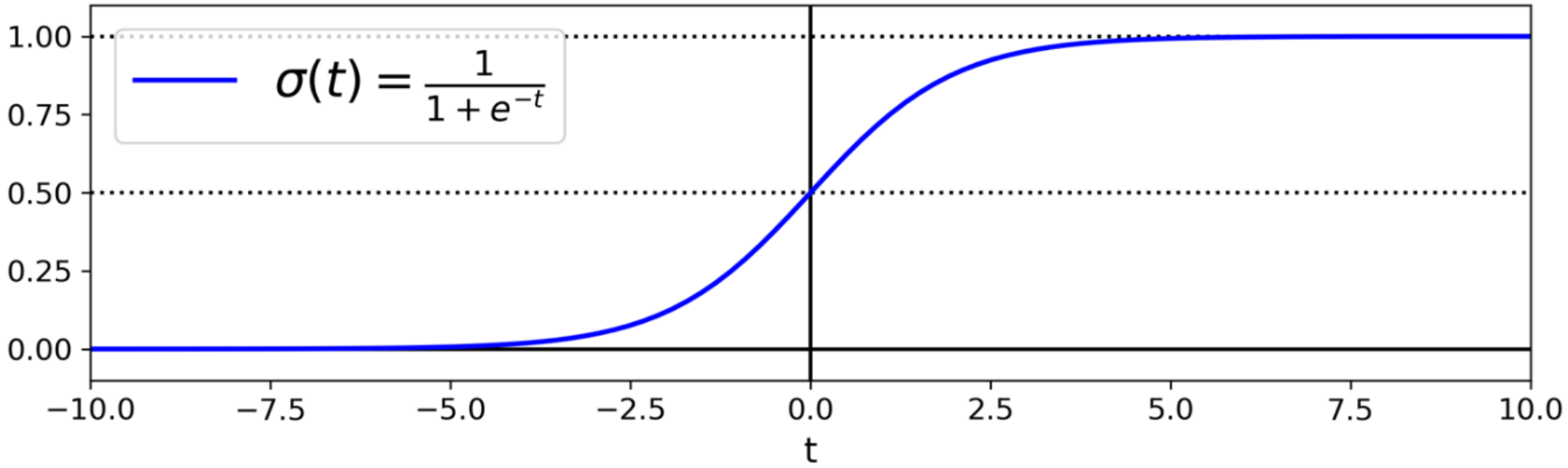
Logistic Regression

- Also known as Logit Regression
- It estimates the probability that an instance belongs to a particular class
- If the estimated probability is greater than 50%, then the model predicts that the instance belongs to Positive class (with label '1'), and otherwise it predicts that it belongs to Negative class (with label '0')
- This makes it a binary classifier
- Logistic Regression model computes a weighted sum of the input features (plus a bias term) and outputs logistic of the result

Logistic Regression...Cont.

- Vectorized Form: $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^{\mathsf{T}}\boldsymbol{\theta})$

- Logistic Function: $\sigma(t) = \frac{1}{1 + e^{-t}}$



Logistic function

Logistic Regression...Cont.

- Logistic Regression Model Prediction: $\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$
- Cost Function of a Single Training Instance: $c(\boldsymbol{\theta}) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1 \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0 \end{cases}$
- Logistic Regression Cost Function (log loss):

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

(Log Loss: Average cost over all training instances)

- Logistic Cost Function Partial Derivatives: $\frac{\partial}{\partial\theta_j}\mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$

Softmax Regression

- It is to generalize *Logistic Regression* to support multiple classes directly
- Also known as *Multinomial Logistic Regression*
- First (Softmax) score $s_k(x)$ for each class k is computed [$\mathbf{s_k(x) = x^T \theta^{(k)}}$]
- Each class has its own dedicated parameter vector $\theta^{(k)}$. All these vectors are typically stored as rows in a parameter matrix Θ
- Then estimates the probability of each class by applying the *softmax function* (also called the normalized exponential) to the scores. The scores are generally called logits or log-odds

K: Number of classes.

s(x): Vector containing scores of each class for the instance x.

$\sigma(s(x))_k$: Estimated probability that the instance x belongs to class k, given the scores of each class for that instance

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

Softmax function

Softmax Regression...Cont.

- Then class with highest probability is predicted using

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^{\mathsf{T}}\mathbf{x}\right)$$

Softmax Regression classifier prediction

- Cost Function (Cross Entropy): $J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$

- y_k^i is the target probability (0 or 1) that the i^{th} instance belongs to class k
- Cross Entropy Gradient Vector for class k: $\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$

Support Vector Machines (SVM)

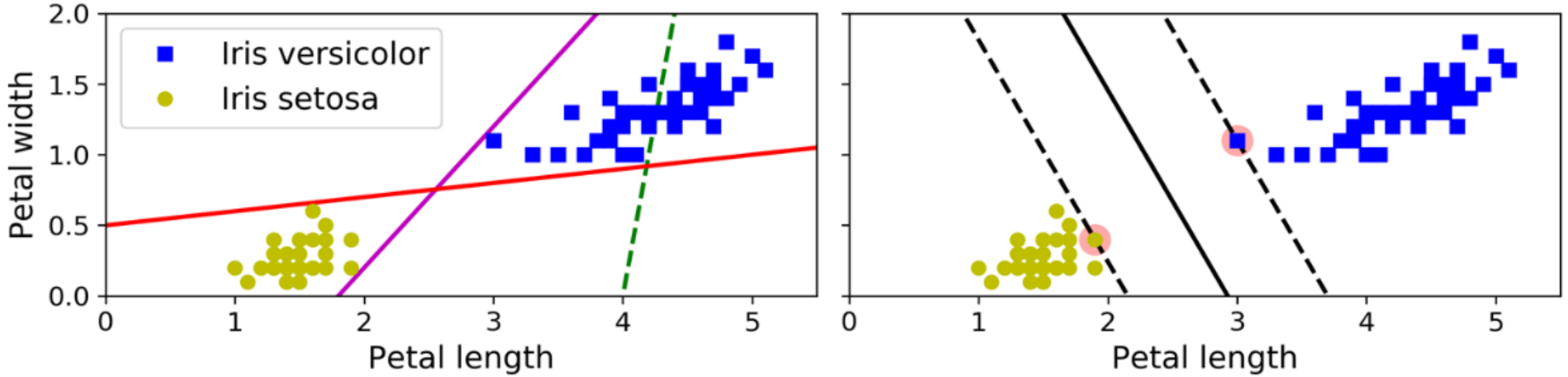
One of the most popular, powerful and versatile Machine Learning models

Supported Tasks

- A SVM can perform both linear and non-linear classification and regression tasks
- It is well suited for classification for small to medium size data set

Linear SVM Classification

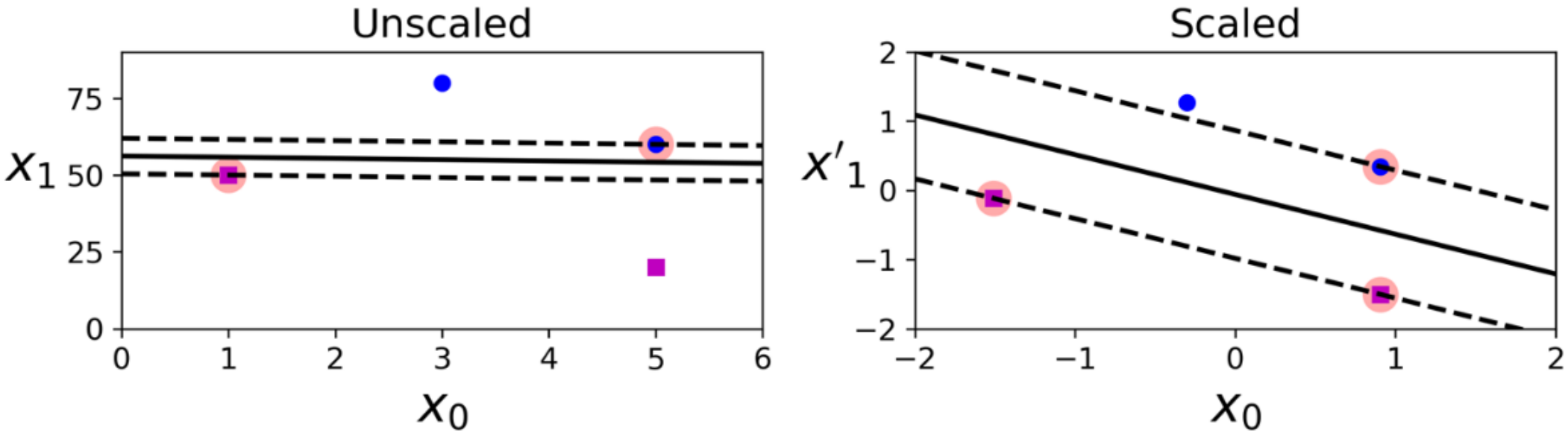
- Separates two classes with a straight line considering they are linearly separable
- Fits decision boundary keeping widest margin between the classes. This is called large margin classification
- Decision boundary is fully determined (or "supported") by the instances located on the edge of the street. These instances are called the support vectors



Decision boundaries (left) and Large margin classification (right)

Linear SVM Classification...Cont.

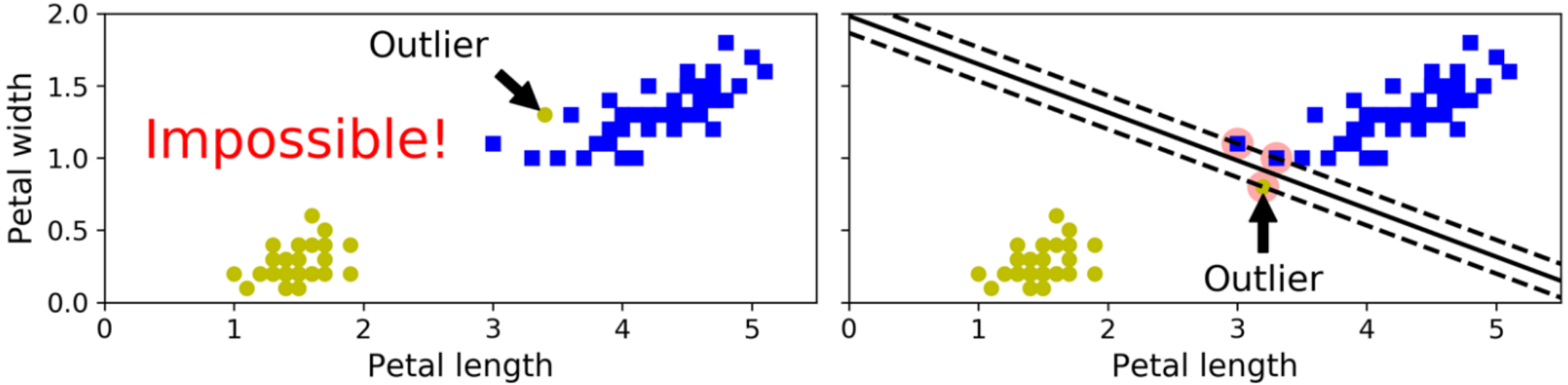
- SVMs are sensitive to the feature scales



Sensitivity to feature scales

Linear SVM Classification...Cont.

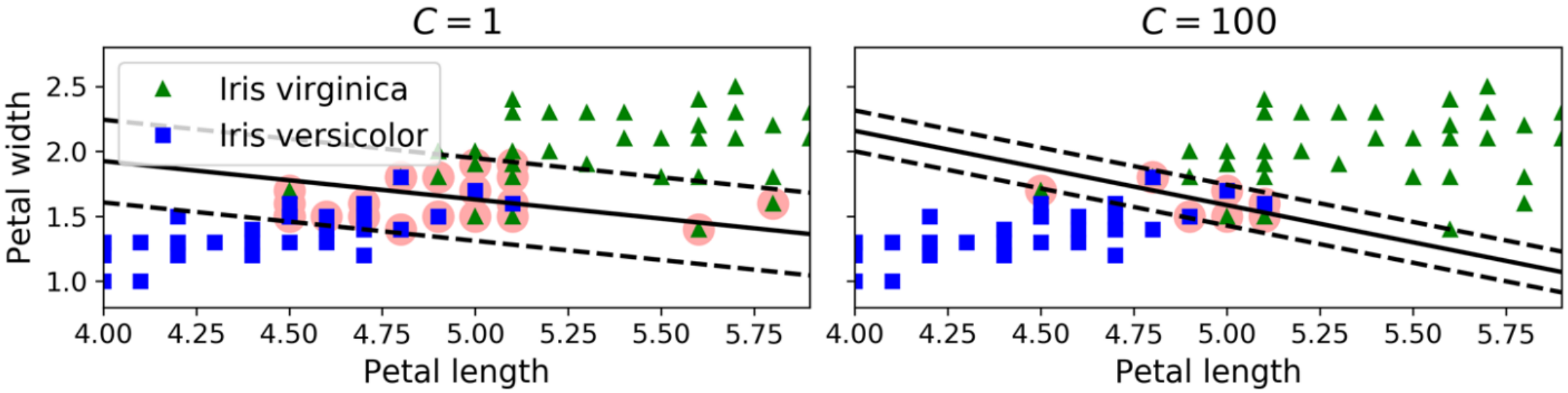
- **Hard Margin Classification:** Imposes no instances to be on the decision line margin
  - It is sensitive to outliers
  - Works if classes are linearly seperable



Hard margin sensitivity to outliers

Linear SVM Classification...Cont.

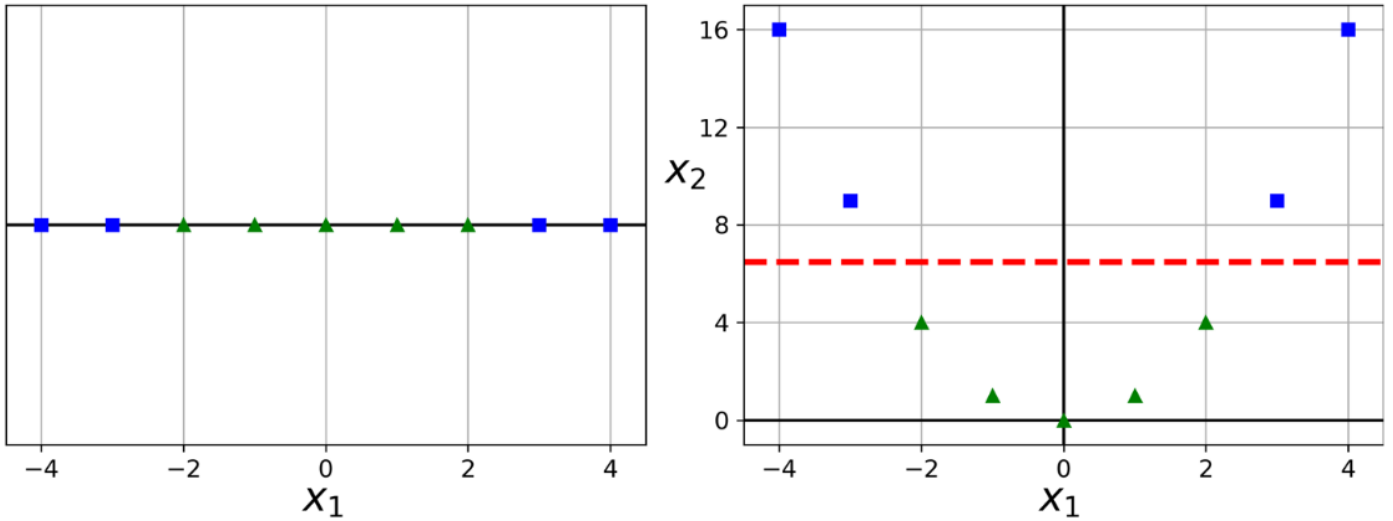
- **Soft Margin Classification:** Finding a good balance between keeping the margin as large as possible and limiting the margin violations



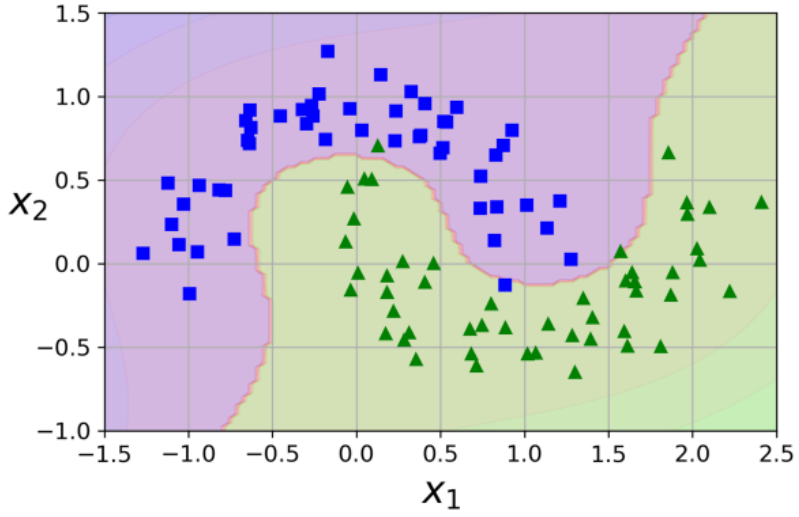
Large margin (left) versus fewer margin violations (right)

Non-linear SVM Classification

- Polynomial features can be added on non-linear data sets to have linearly separable classes



Adding features to make a dataset linearly separable

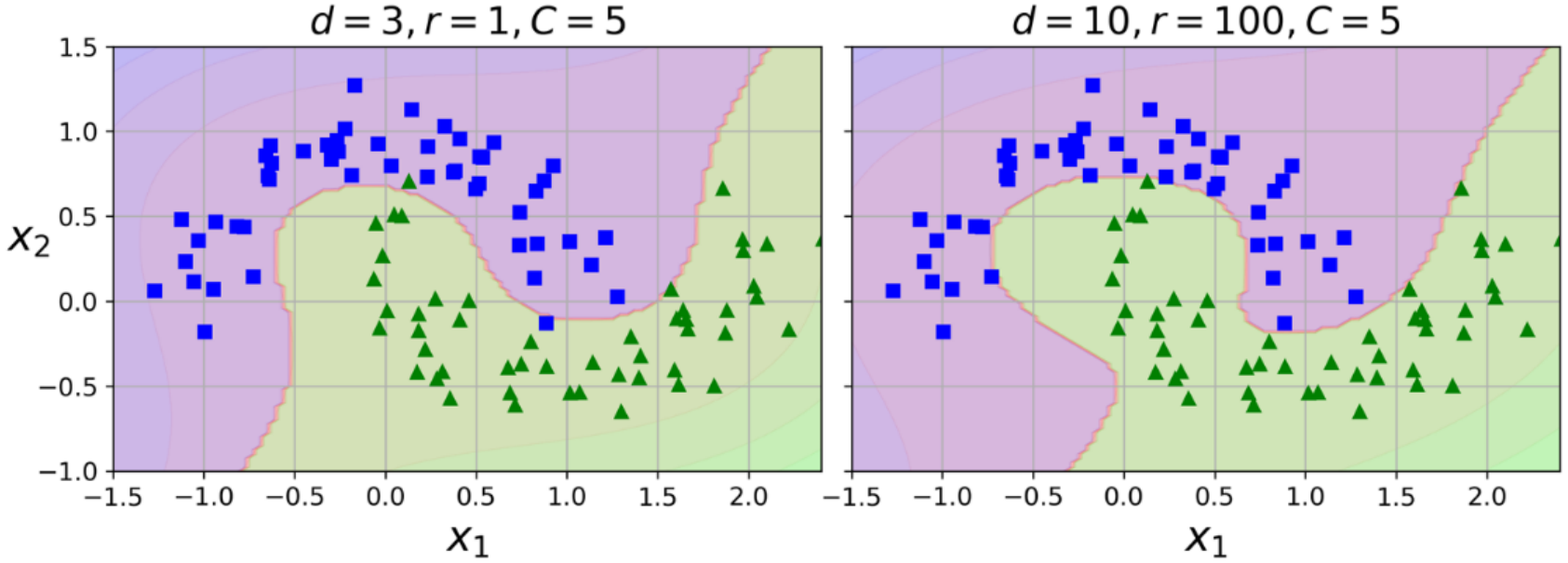


Linear SVM classifier using polynomial features

Non-linear SVM Classification…Cont.

Polynomial Kernel

- SVM kernel internally supports polynomial features without being explicitly added as features in data set. This is called *kernel trick*.



SVM classifiers with a polynomial kernel

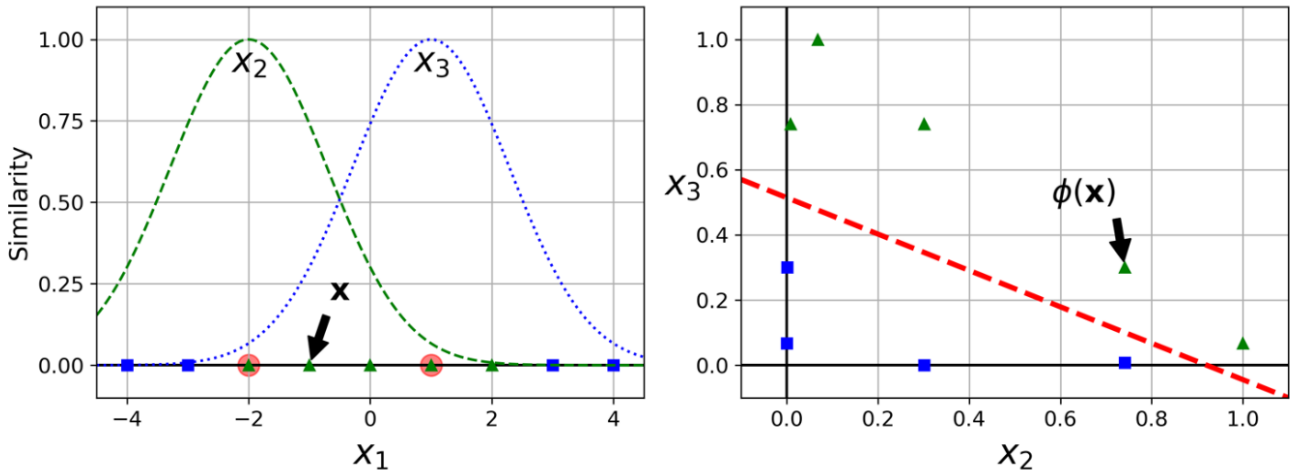
PLACE HOLDER

Non-linear SVM Classification…Cont.

Similarity Features

- Adds features computed using a similarity function, which measures how much each instance resembles a particular landmark
- Gaussian Radial Basis Function (RBF) is used as similarity function

$$\phi_\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma \| \mathbf{x} - \ell \|^2\right)$$



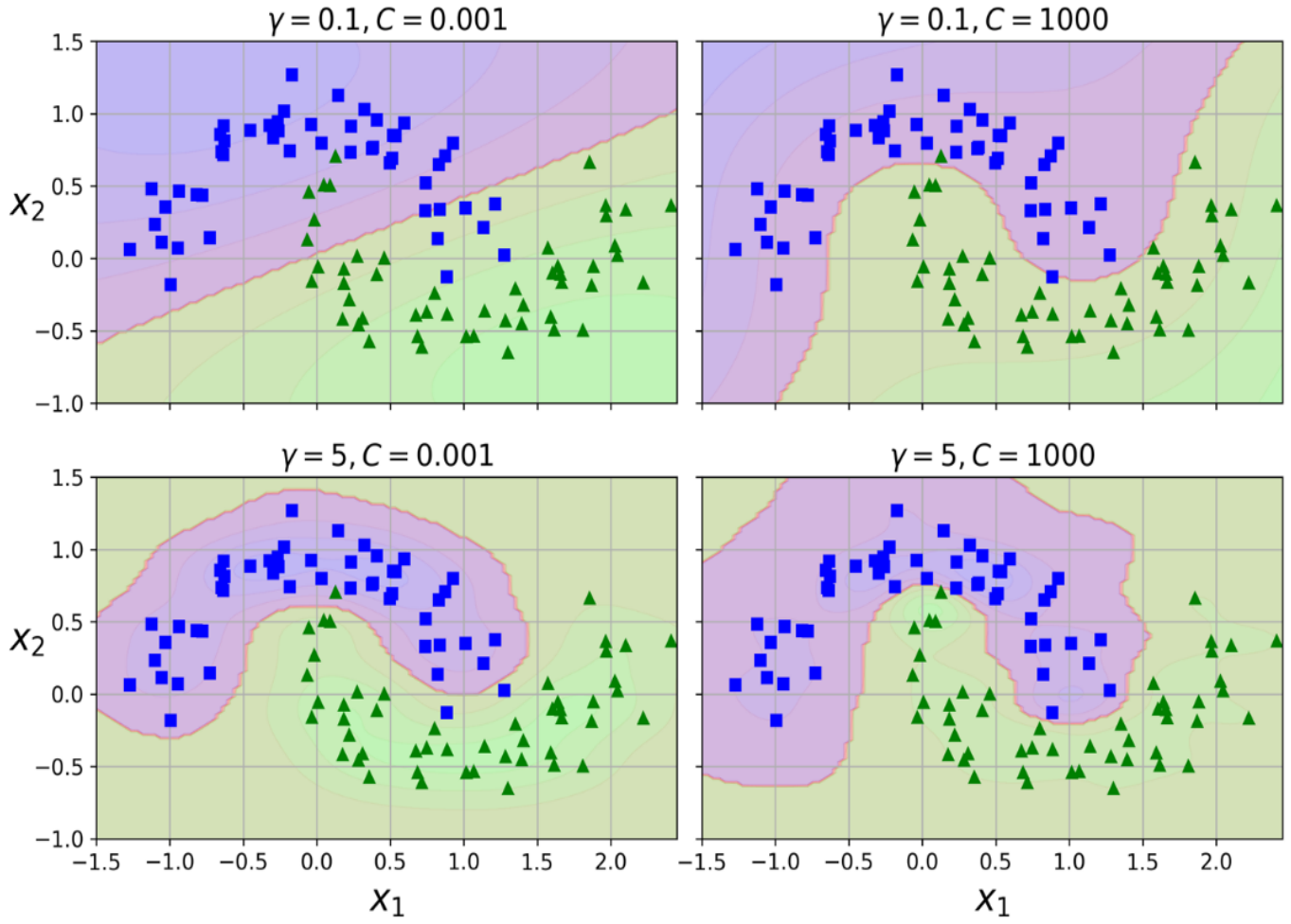
Non-linear SVM Classification...Cont.

Gaussian Radial Basis Function (RBF) Kernel

- SVM kernel internally supports similarity features without being explicitly added as features in data set. This is called also a *kernel trick*.

Hyperparameters:

- gamma (γ): Decision boundary ends up smoother with smaller gamma values
- C: Decides strength (inversely proportional to its value) of regularization



SVM classifiers with an RBF kernel

Non-linear SVM Classification...Cont.

Rule of Thumb for Deciding SVM Kernel

- Trying linear kernel first (LinearSVC is much faster than SVC(kernel="linear"))
- Trying Gaussian RBF kernel if data set is not large
- Cross-validation and grid search for hyperparameter tuning based on the availability of time and computing power

Non-linear SVM Classification...Cont.

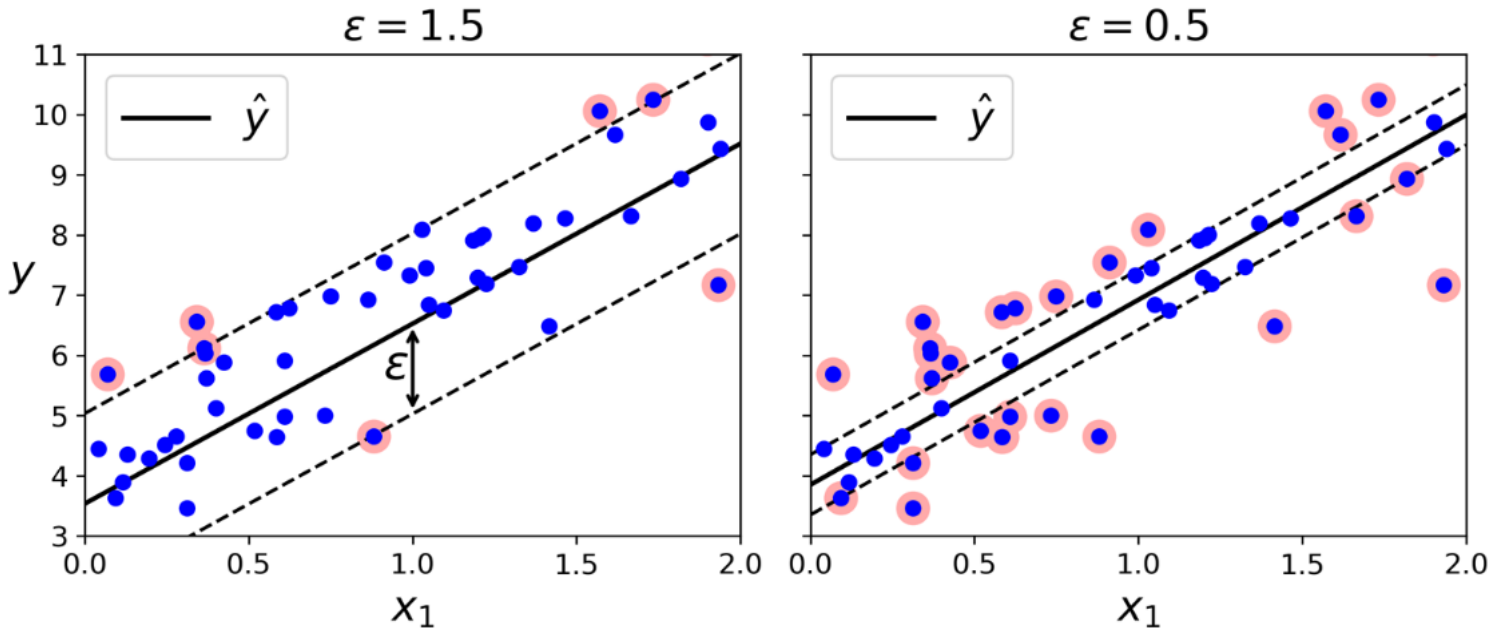
Computational Complexity of Classifiers

Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes

Comparison of Scikit-Learn classes for SVM classification

SVM Regression

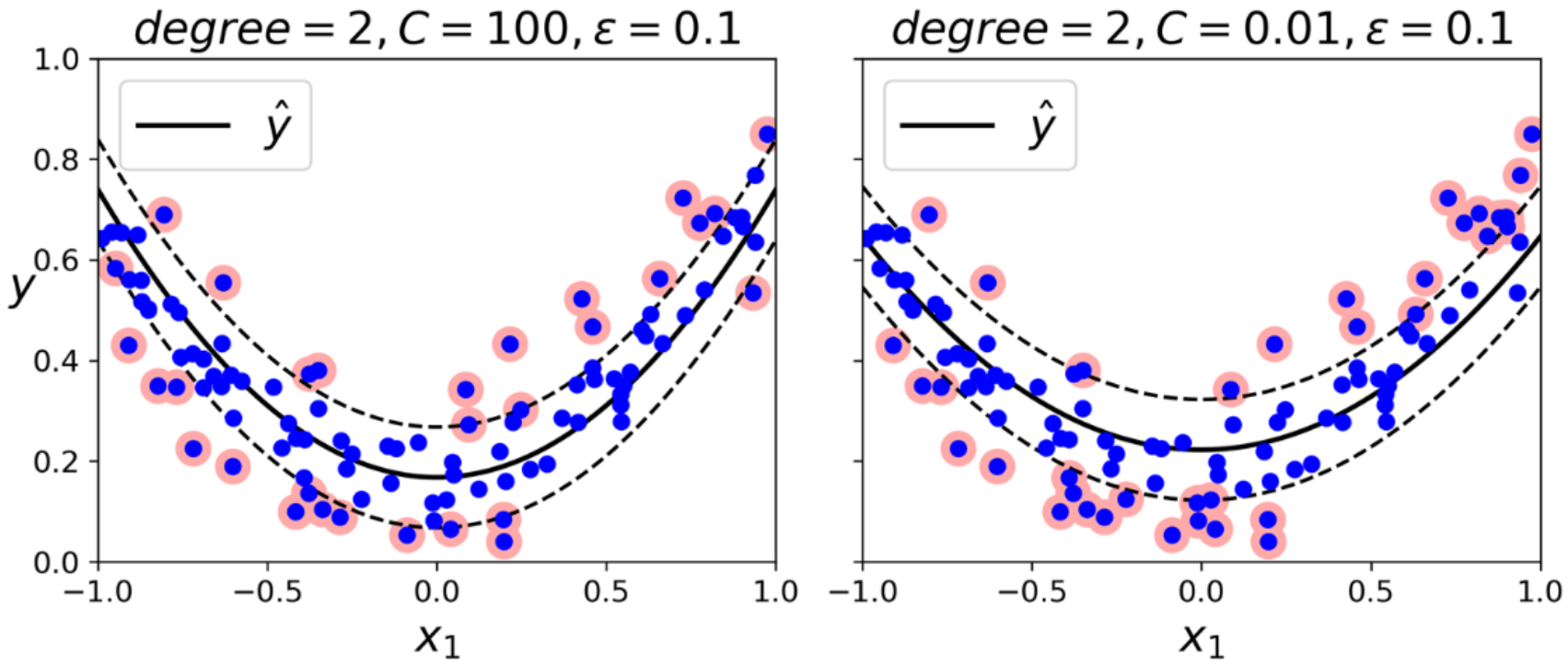
- SVM also supports linear and non-linear regression
- Fits as many instances as possible on the street while limiting margin violations



Linear SVM Regression

Non-linear SVM Regression

- Kernelized SVM supports non-linear regression



Hyperparameters:

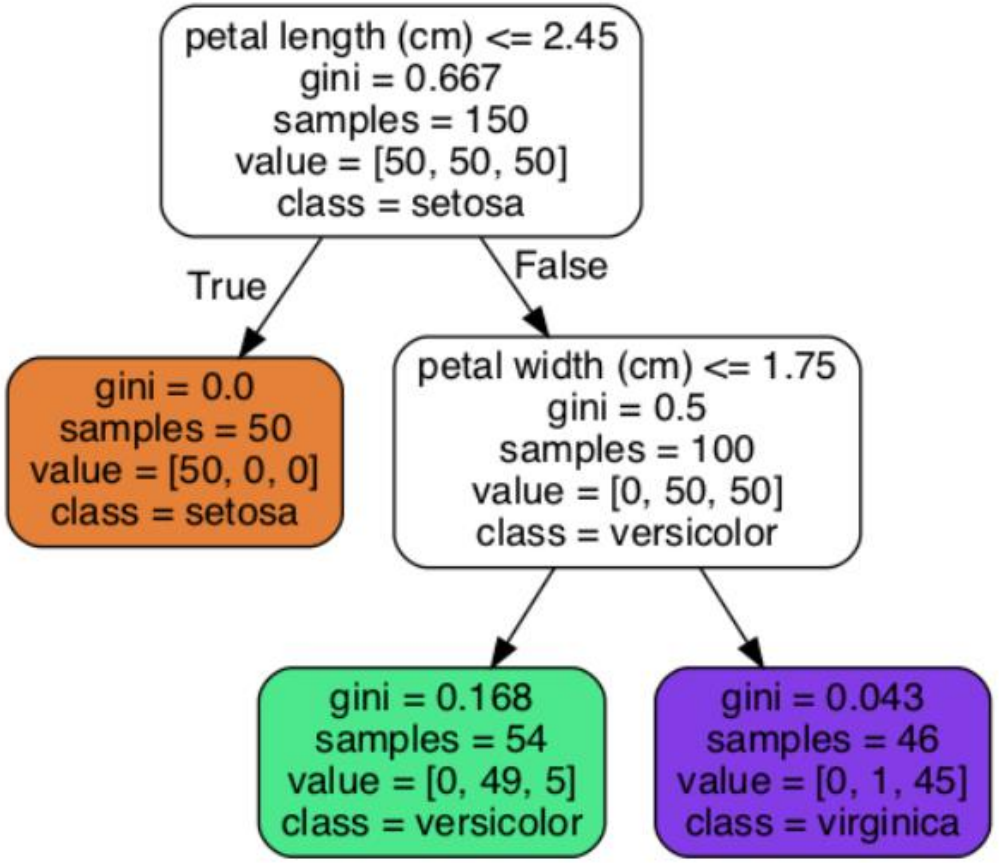
- degree: Polynomial feature
- C: Decides strength (inversely proportional to its value) of regularization
- ε (epsilon): Width of the margin

Decision Trees

A versatile Machine Learning algorithm that can perform both classification and regression tasks, and even multioutput tasks. It's a powerful algorithm capable of fitting complex datasets.

Introduction to Decision Trees

- Performs both classification, regression and multi-output tasks.
- Powerful algorithm to fit non-linear data set.
- Standard Implementations are DecisionTreeClassifier, DecisionTreeRegressor, RandomForestClassifier, RandomForestRegressor.
- Scaling is not required.
- It is a binary classifier (ID3 can have node with > 2 children)
- It is white box model (decisions are interpretable).



A Decision Tree*

*Decision Tree was built with DecisionTreeClassifier with max_depth = 2

Gini Impurity

- Gini attributes measure a node's impurity

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

$$H_i = -\sum_{k=1}^{n} p_{i,k} \log(p_{i,k})$$

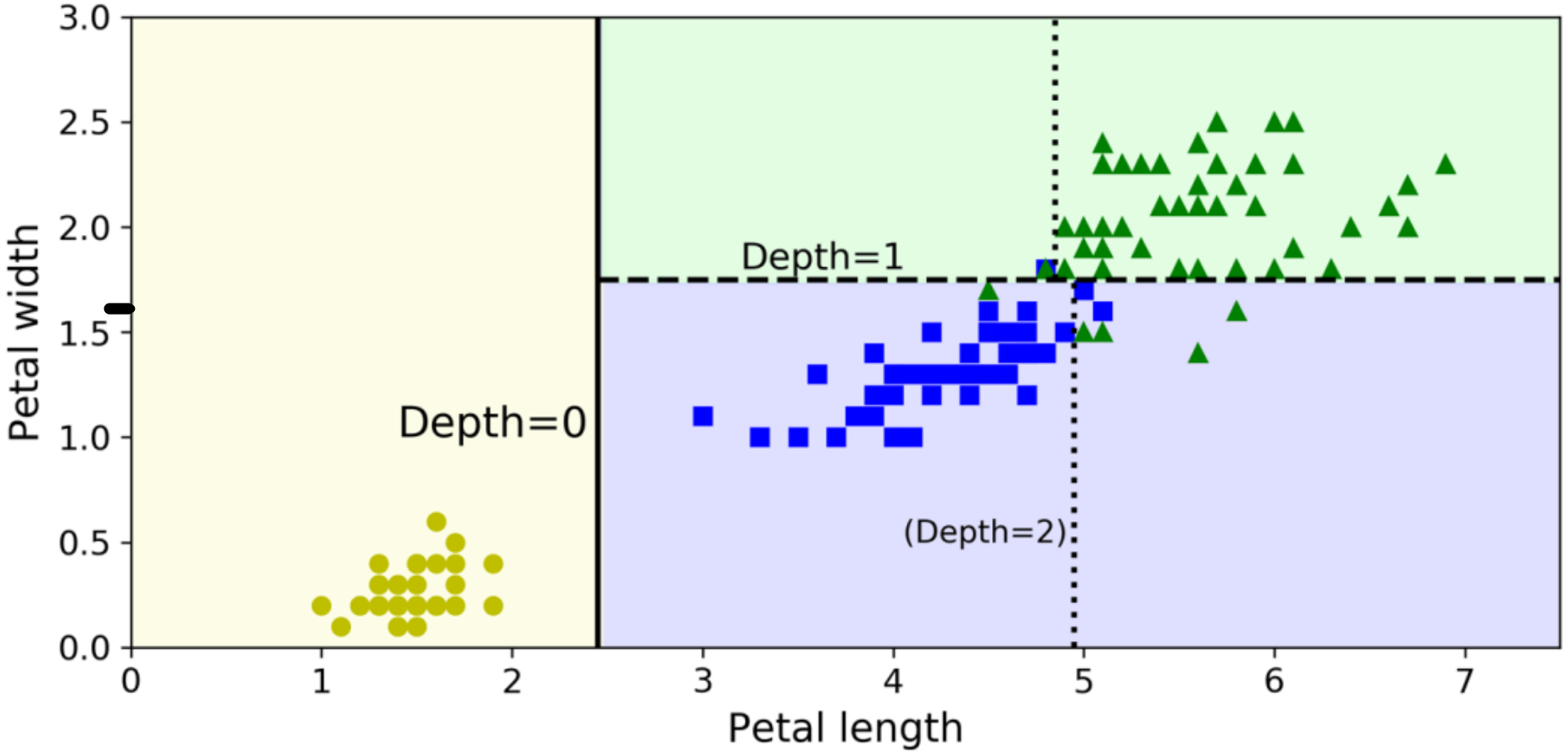
Entropy

- Gini value zero (0) of a node indicates that it is "pure" (all instances in that node belong to same class)

G_i: Gini impurity of i^{th} node

$p_{i,k}$: Ratio of k instances among training instances in i^{th} node

Decision Tree Decision Boundaries



A Decision Tree decision boundaries

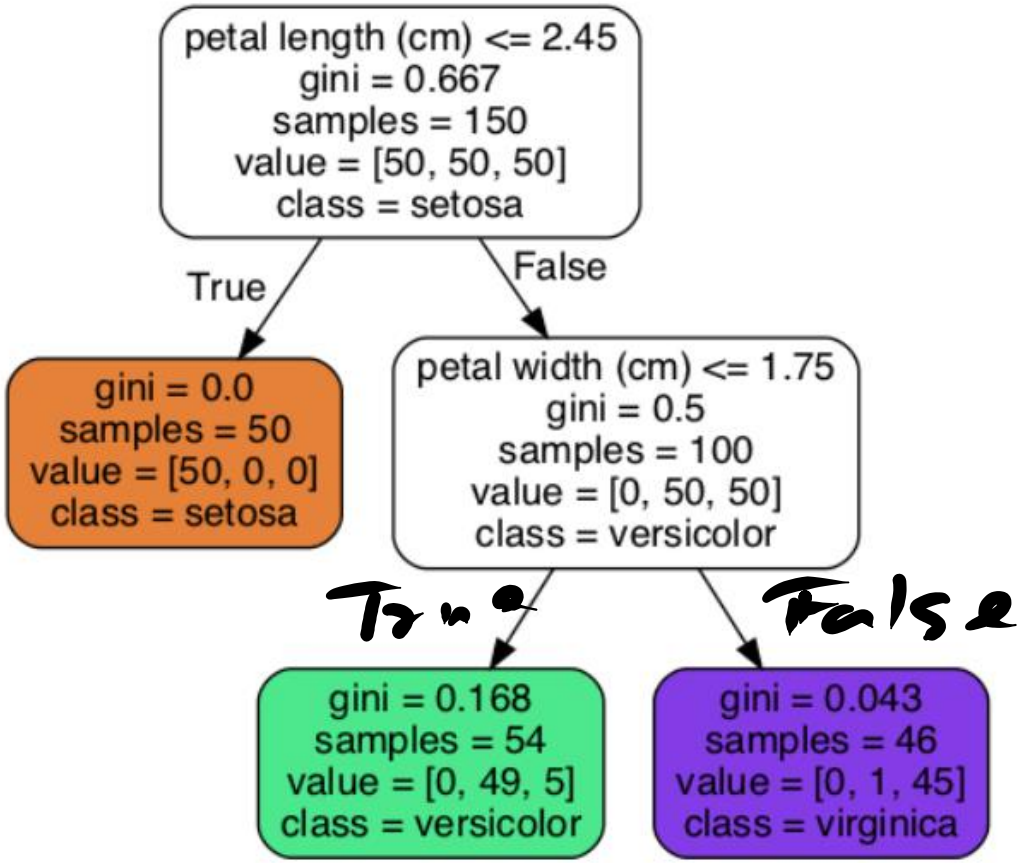
Estimating Class Probabilities

Estimating the probability that an instance belongs to a particular class *k* if a flower's petals are 5 cm long and 1.5 cm wide.

$$p_1 = \frac{0}{0+49+5} = \frac{0}{54} = 0$$

$$p_2 = \frac{49}{0+49+5} = \frac{49}{54} =$$

$$p_3 = \frac{5}{0+49+5} = \frac{5}{54} =$$



Classification and Regression Tree (CART) Algorithm

- First splits the training set into two subsets using a single feature *k* and a threshold t_k (e.g., “petal length ≤ 2.45 cm”).
- Then searches for the pair (k, t_k) that produces the purest subsets (weighted by their size).
- It tries to minimize the cost function (refer next slide).
- Once training set is split in two successfully, it splits the subsets using the same logic, then the sub-subsets, and so on, recursively.
- It stops recursing once it reaches maximum depth (*max_depth* hyperparameter*), or if it can't find a split that will reduce impurity.

*Other hyperparameters that control additional stopping conditions are *min_samples_split*, *min_samples_leaf*, *min_weight_fraction_leaf* and *max_leaf_nodes*.

Classification and Regression Tree (**CART**) Algorithm...Cont.

Cost Function for Classification:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where} \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$$

CART algorithm is a greedy algorithm. It greedily searches for an optimum split at the top level, then repeats the process at each subsequent level. It does not check whether the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a solution that's reasonably good but not guaranteed to be optimal.

Computation Complexity

- Making predictions requires traversing the Decision Tree from the root to a leaf.
- Decision Trees generally are approximately balanced.
- Traversing the Decision Tree requires going through roughly $O(\log_2(m))$ nodes.
- Overall prediction complexity is $O(\log_2(m))$ (feature independent).
- Predictions are fast even with large training set.

Gini Impurity or Entropy Impurity Measure?

- DecisionTreeClassifier support both.
  - DecisionTreeClassifier(criteria="gini") for use Gini
  - DecisionTreeClassifier(criteria="entropy") for use Entropy
- Gini is default impurity measure in DecisionTreeClassifier.
- Most of the time, both lead to similar trees.
- Gini impurity is slightly faster to compute, and hence is default in DecisionTreeClassifier.

Regularization Hyperparameters

- Decision Trees make very few assumptions (on linear or non-linear) about the training data.
- It may overfit training data if left unconstrained.

DecisionTreeClassifier's Regularization Hyperparameters

Hyperparameter	Description
max_depth [Default: None]	Depth of tree
max_leaf_nodes [Default: None]	Maximum number of leaf nodes
max_features [Default: None]	Maximum number of features that are evaluated for splitting at each node
min_samples_split [Default: 2]	Minimum number of samples a node must have before it can be split
min_samples_leaf [Default: 1]	Minimum number of samples a leaf node must have
min_weight_fraction_leaf [Default: 0.0]	Same as min_samples_leaf but expressed as a fraction of total number of weighted instances

Increasing min_* hyperparameters or reducing max_* hyperparameters will regularize the model.

PLACE HOLDER

Regularization Hyperparameters…Cont.

- **Non-parametric Model:** If a Decision Tree is left unconstrained (parameters is not determined prior to training), the tree structure will adapt itself to the training data, fitting it very closely resulting overfitting it.
- **Parametric Model:** A model having predefine number of parameters, so its degree of freedom is limited, reducing the risk of overfitting (but increasing the risk of underfitting).
- **Pruning:** Training the Decision Tree without restrictions, then pruning (deleting) unnecessary nodes. <More details to come>

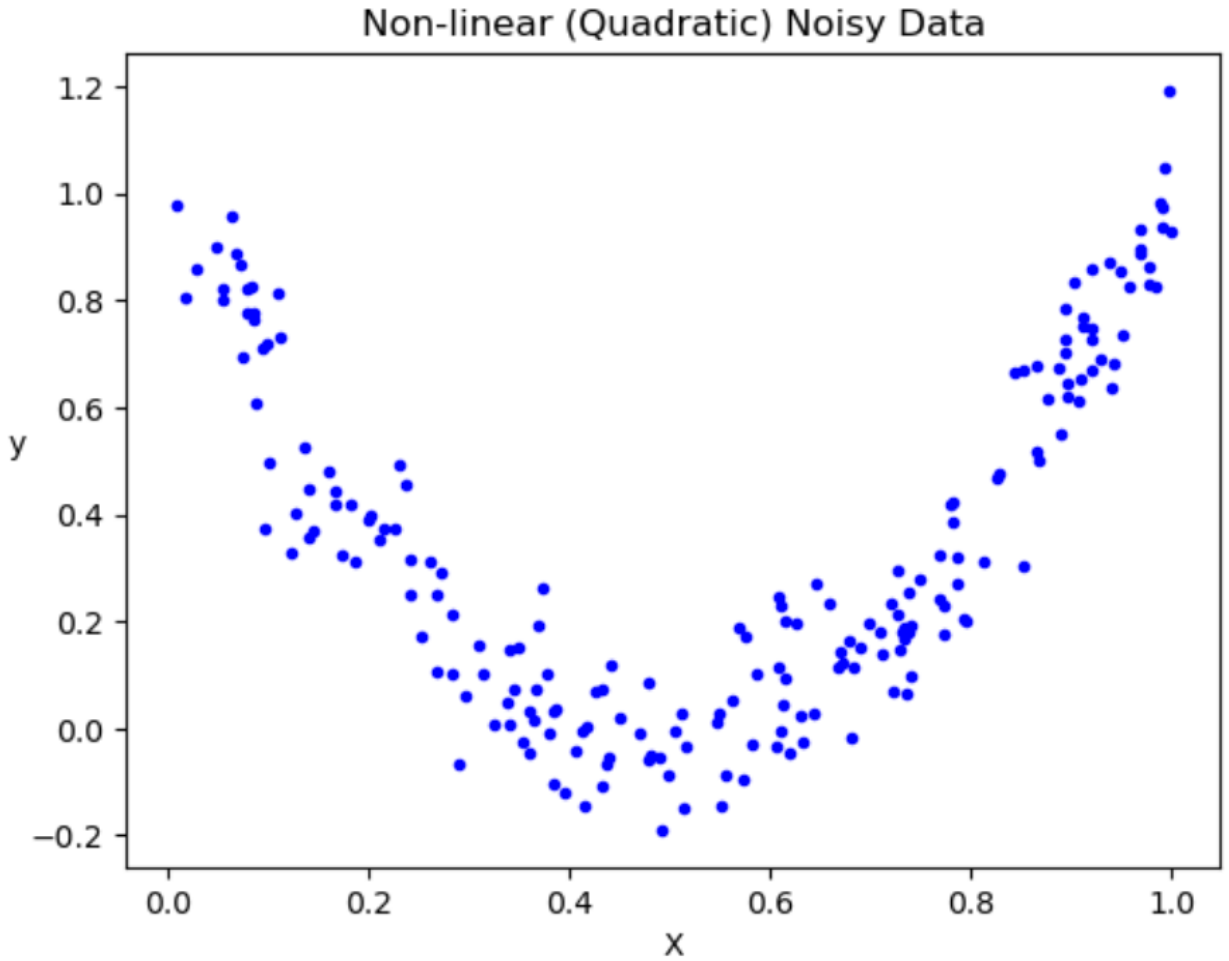
Differences between CART-based & ID3-based Decision Tree

	CART Decision Tree	ID3 Decision Tree
1.	Performs both classification and regression task.	Performs only classification task.
2.	It's a binary tree.	It may have more than two child nodes.
3.	Works on both categorical and numerical (discrete and continuous) data.	Works only on categorical data.
4.	Supports both Gini and Entropy as impurity measure.	Supports only Entropy as impurity measure.
5.	A feature can be used more than one time at any depth as condition to split on	A feature can only be used once across tree as condition to split on
6.	Popular and widely used.	Not much popular and used.

Decision Tree Regression

- Performs regression tasks.
- One of the implementations is DecisionTreeRegressor in Scikit-Learn.

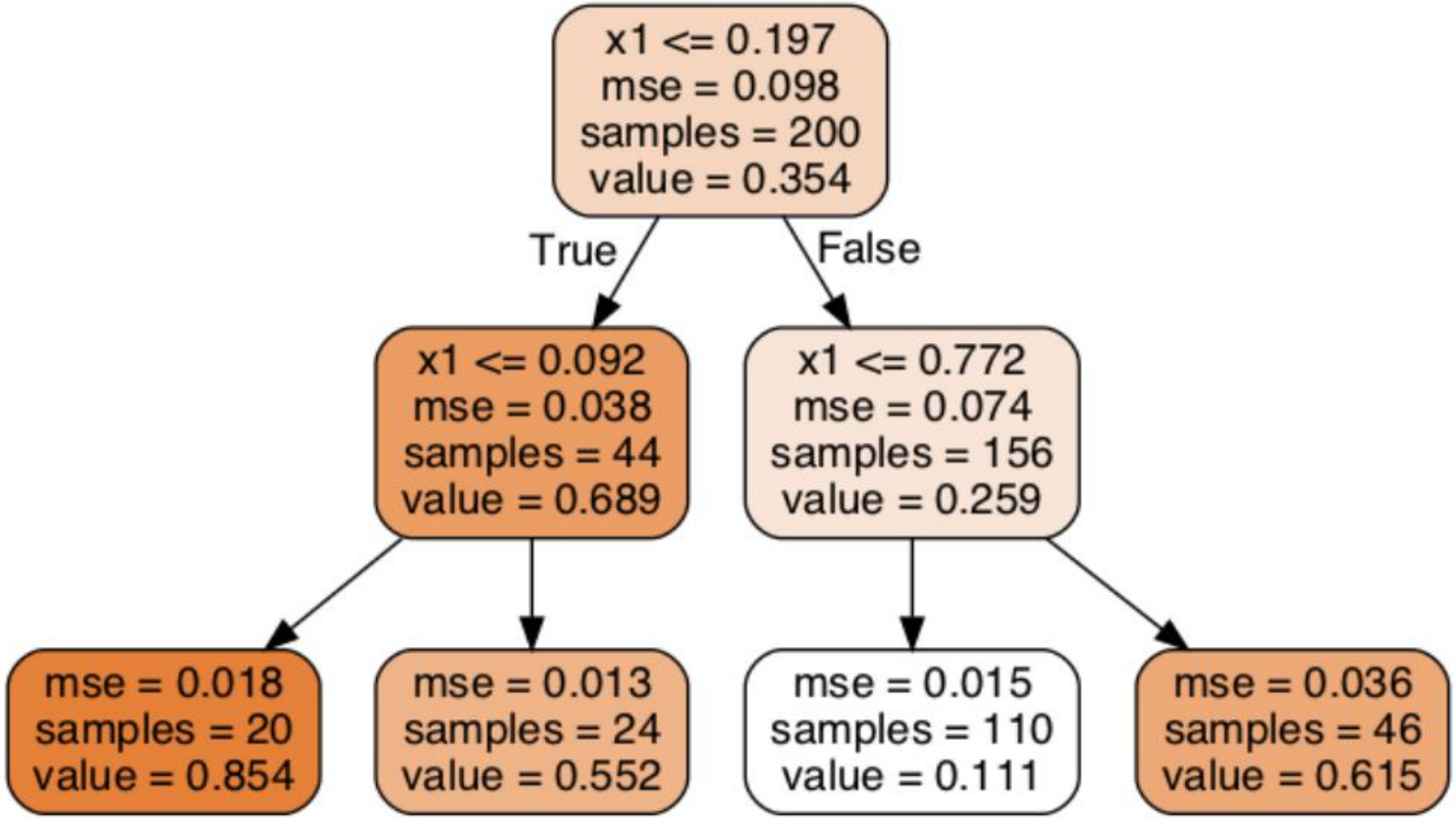
Noisy Data Sample for Model to Fit On



$y = 4*(x-0.5)^2 +$ Gaussian noise

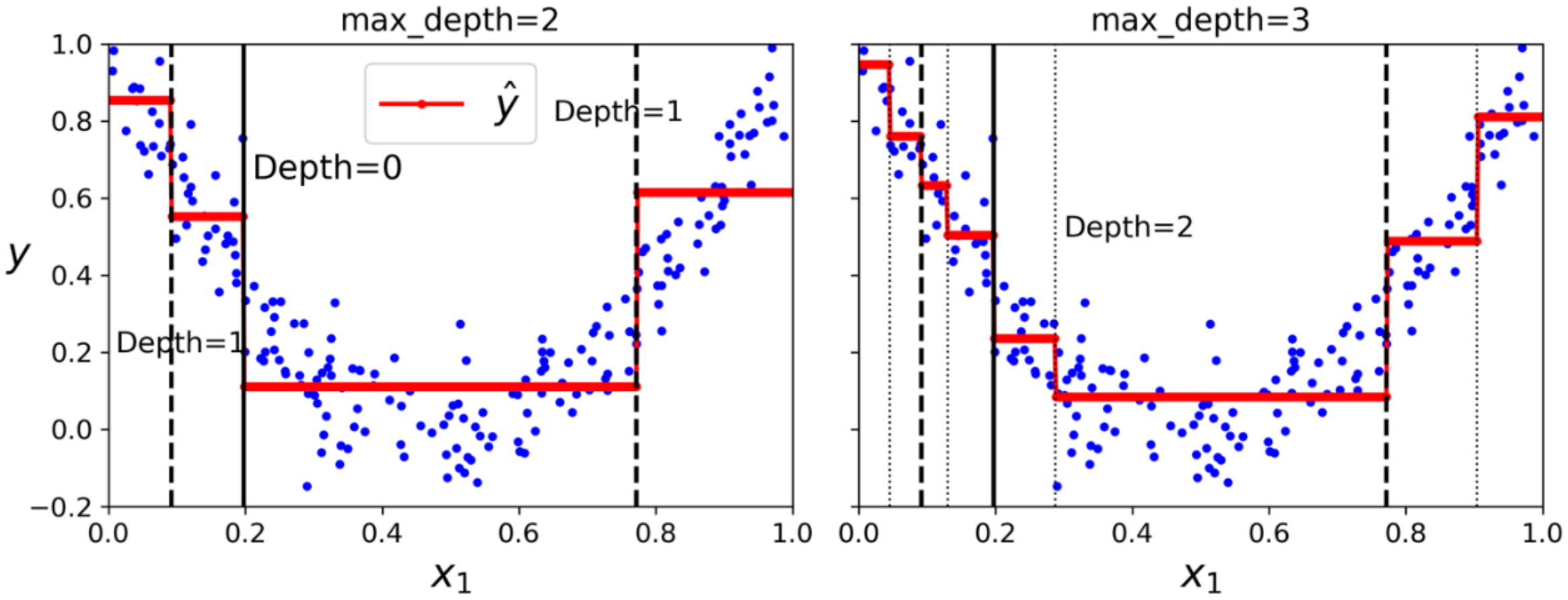
Noisy data where $y = 4*(x-0.5)^2$ + <Gaussian Noise>

DecisionTreeRegressor Model



A decision tree for regression with depth 2

Decision Tree Regression



A Decision Tree decision boundaries

Cost Function for Regression in CART Algorithm

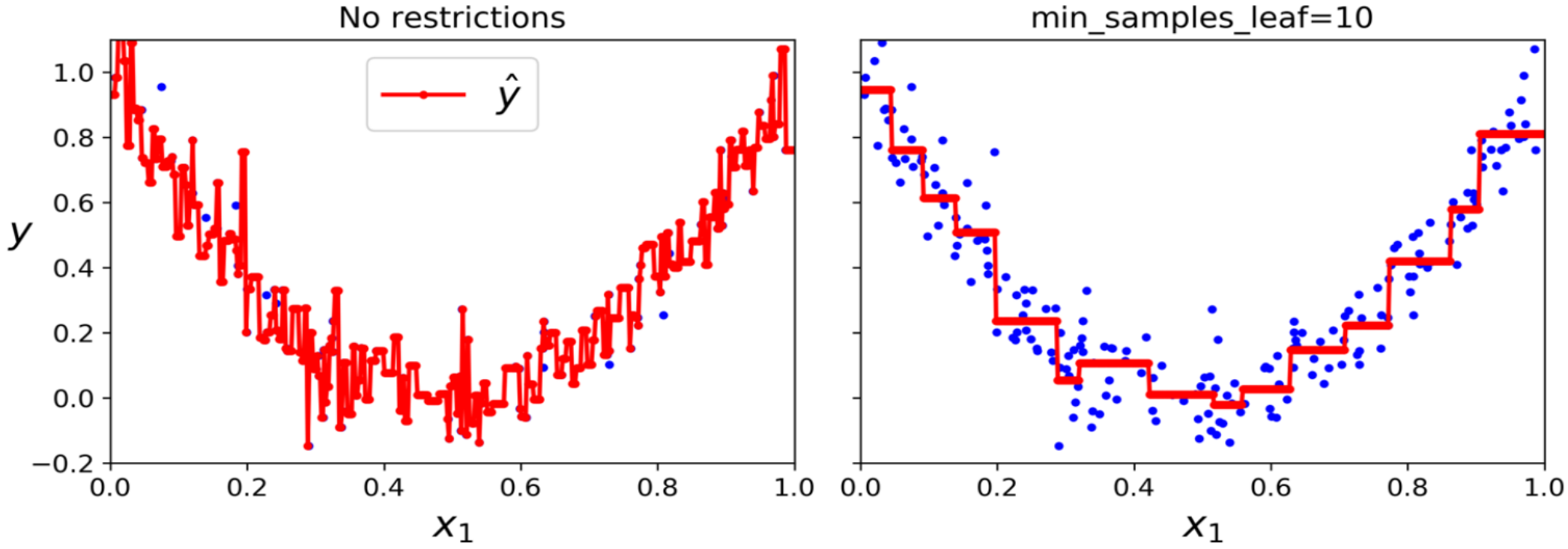
- Splits training set in a way that minimizes the MSE (cost function).
- Cost Function for Regression:

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}$$

$$\text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Regularizing Decision Tree Regressor

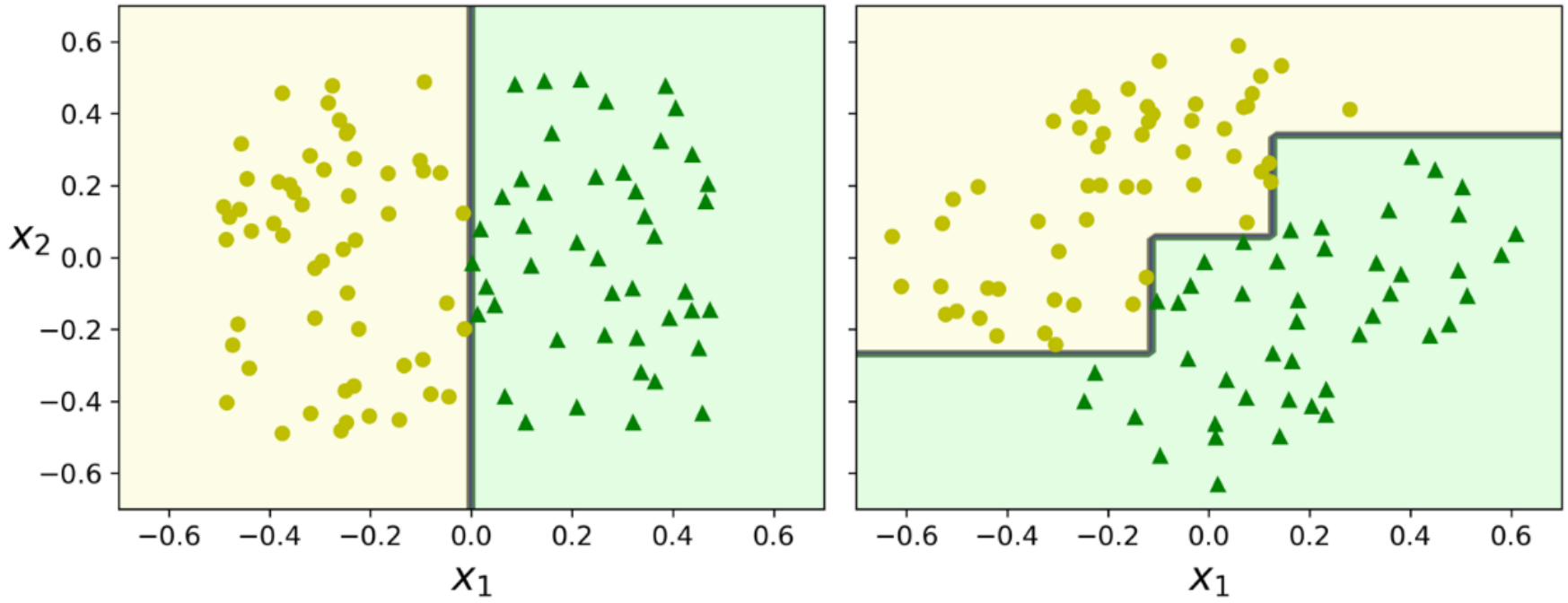
- Decision Tree Regressor, too, can overfit itself to the training data.
- Applying appropriate regularization may result a reasonable model.



A regularized Decision Tree regressor

Characteristics of Decision Tree

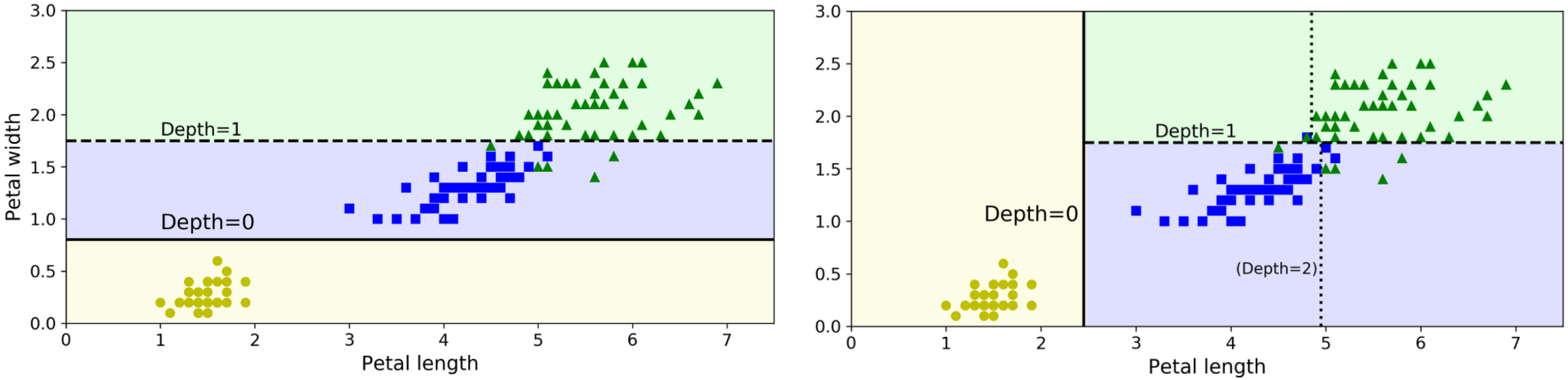
- Simple to use, easy to understand and interpret, and powerful.
- It develops orthogonal decision boundaries (perpendicular to an axis) making it sensitive to training data rotation.



Sensitivity to training set rotation

Characteristics of Decision Tree...Cont.

- It's sensitive to small variation in training data.



Model trained after removing widest (petals 4.8 cm long and 1.8 cm wide) versicolor from iris training set (left), and previous model trained with all training data (right)

- As CART Decision Tree (in Scikit-Learn) is stochastic (unless *random_state* is set), different models may get created even on same training data. Ensembling such as Random Forest addresses this by averaging predictions over many trees.

Ensemble Learning & Random Forests

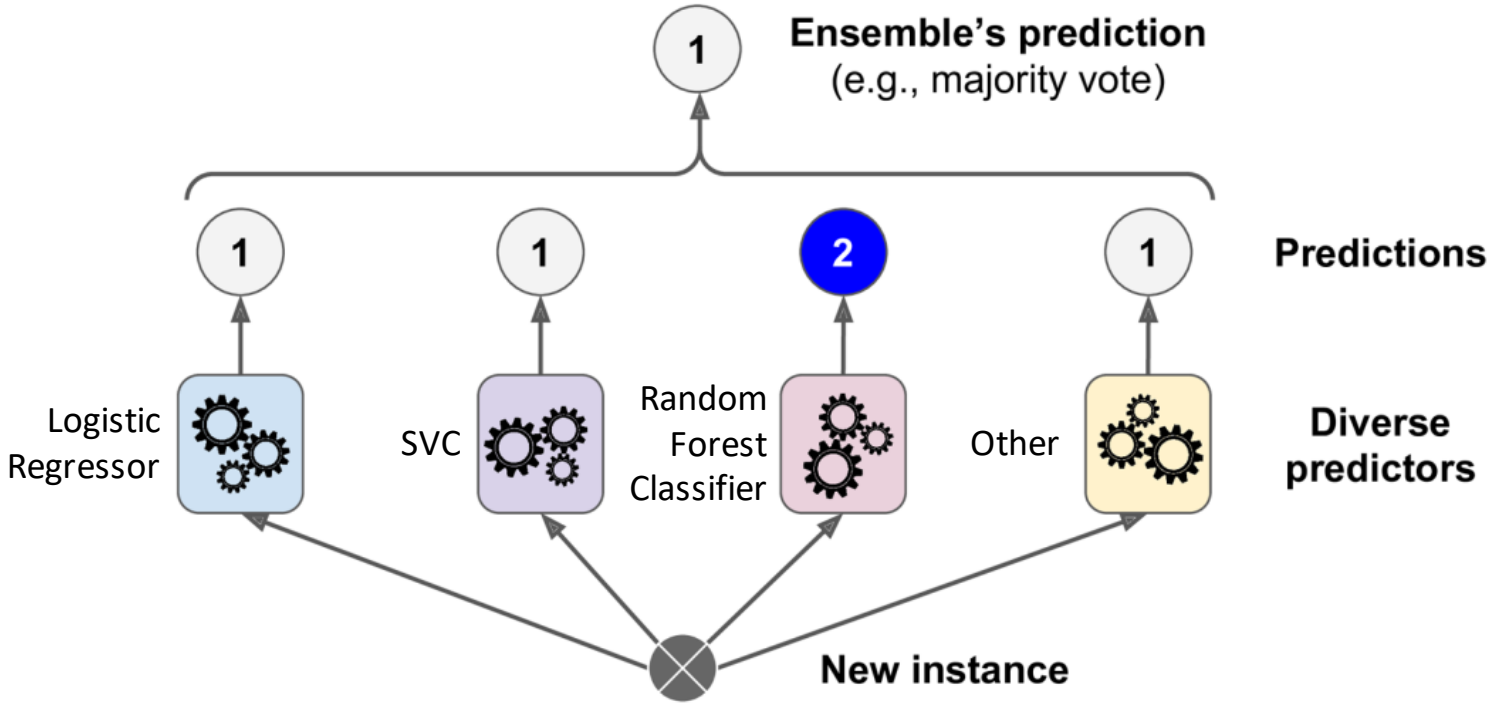
Applying the power of aggregated predictions of a group of predictors

Ensemble Learning

- A technique of aggregating predictions of a group of predictors is called Ensemble Learning.
- Often yields better predictions than with best individual predictor.
- An Ensemble Learning algorithm is called Ensemble method such as *Bagging*, *Boosting* and *Stacking* and *Random Forest*.
- Aggregation strategy could be most votes, average, etc.
- Ensemble methods work best when predictors are as independent from one another as possible. One way to get diverse classifiers is to train them using very different algorithms.

Voting Classifiers

- **Hard Voting**: Aggregating the predictions of each classifier and predict class that gets the most votes.



Voting Classifiers...Cont.

One of the implementations of Voting classifier is VotingClassifier in Scikit-Learn.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Considering moon data set for this classification task

Voting Classifiers…Cont.

Looking at each classifier's accuracy on the test set:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

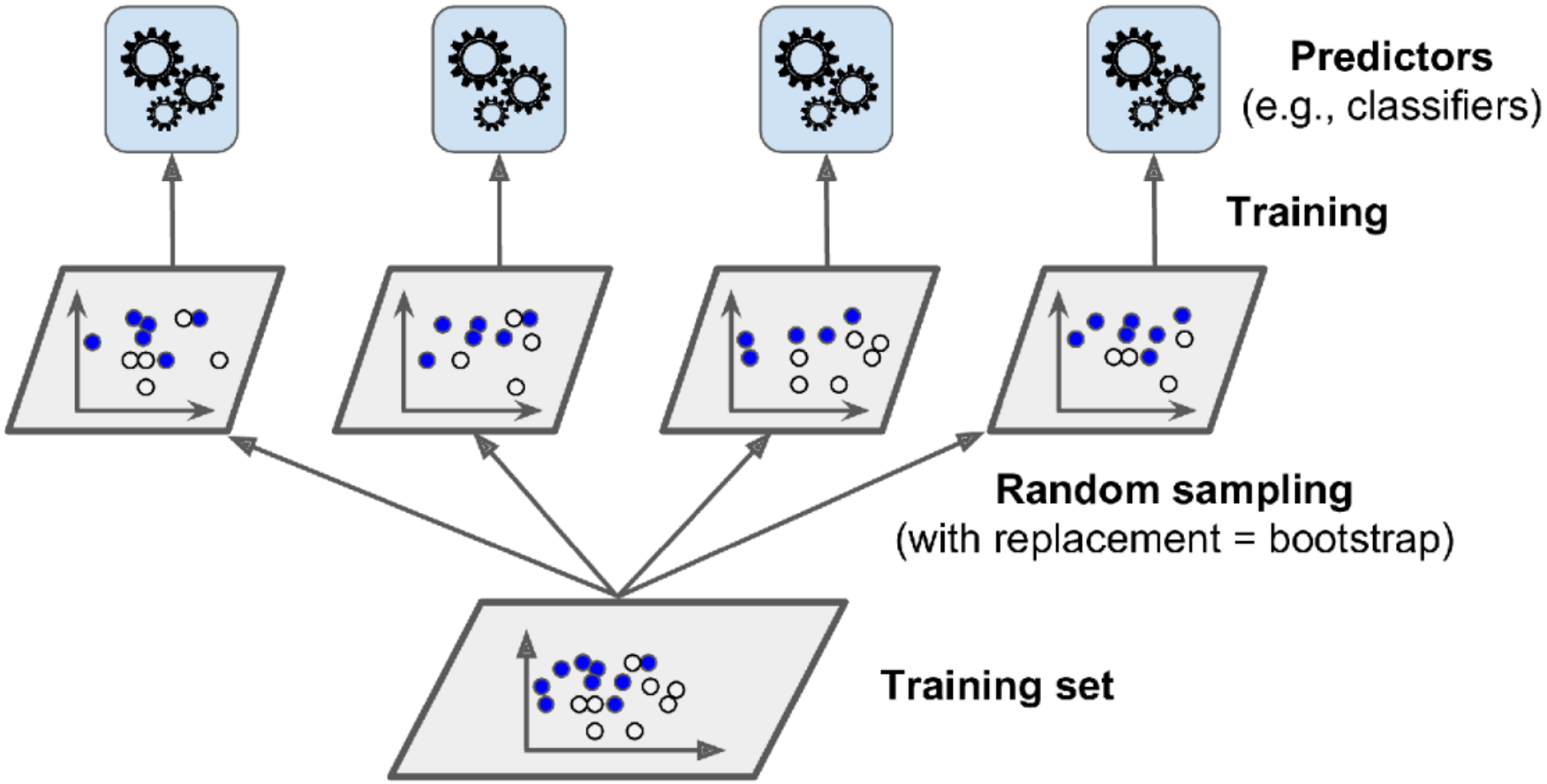
Voting Classifiers…Cont.

- **Soft Voting**:
  - Predicting the class with the highest class probability, averaged over all the individual classifiers.
  - Achieves higher performance than hard voting as it gives more weight to highly confident votes.
  - This can be implemented by setting *voting="soft"* on the VotingClassifier's constructor.

Bootstrap Aggregating (Bagging) & Pasting

- Uses same training algorithm for every predictor and train them on different random subsets (sampling) of the training set.
- When sampling is performed with replacement, this method is called *Bagging*, and
- When sampling is performed without replacement, it is called *Pasting*.
- Ensemble makes prediction for a new instance by simply aggregating predictions of all predictors.
  - Most frequent prediction for classification, or
  - Average for regression
- Ensemble has a similar bias but a lower variance than a single predictor

Bootstrap Aggregating (Bagging) & Pasting…Cont.



Bagging and pasting involves training several predictors on different random samples of the training set

Bootstrap Aggregating (Bagging) & Pasting…Cont.

- One of the implementations for both Bagging and Pasting classifier is BaggingClassifier (and BaggingRegressor for regressor) in Scikit-Learn.

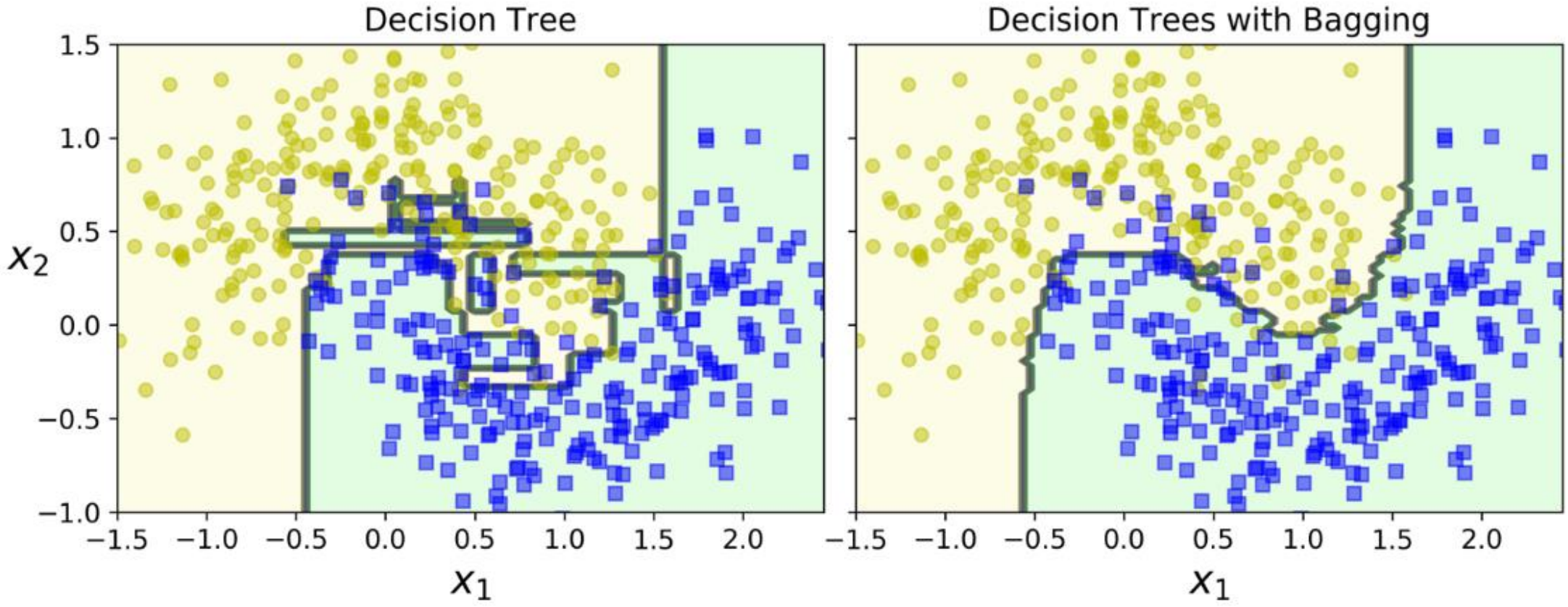
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Default voting: Soft; Dataset: Moon

Bootstrap Aggregating (Bagging) & Pasting…Cont.

- Better generalization than the single Decision Tree's prediction



A single Decision Tree (left) versus a bagging ensemble of 500 trees (right)
Ensemble has a comparable bias but a smaller variance

Out-of-Bootstrapping-Aggregation (Out-of-Bagging) Evaluation

- Training instances that are not sampled are called Out-of-Bag (OOB) instances.
- For a predictor, these unseen Out-of-Bag instances can be used to evaluate the predictor on those instances without the need for a separate validation set.
- Ensemble's overall performance can be evaluated by averaging out OOB evaluation of each predictor.

```
>>> bag_clf = BaggingClassifier(
...     DecisionTreeClassifier(), n_estimators=500,
...     bootstrap=True, n_jobs=-1, oob_score=True)
...
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
0.90133333333333332
```

Random Patches and Random Subspaces

- Like random instance sampling, random features sampling is also supported by BaggingClassifier.
- Each predictor will be trained on a random subset of features.
- The hyperparameters are max_features and bootstrap_features.
- Sampling both training instance and features are called Random Patches method.
- It's useful especially when dealing with high-dimensional input such as images.
- Sampling just features keeping all training instances is called Random Subspaces method.

Random Forests

- An ensemble of Decision Trees, generally trained via the bagging method
- More convenient and optimized than BaggingClassifier with DecisionTreeClassifier as predictors

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Random Forests...Cont.

- Hyperparameters:
  - Has all the hyperparameters of a DecisionTreeClassifier (to control how trees are grown)
  - Has all the hyperparameters of a BaggingClassifier to control the ensemble itself
- Implements extra randomness when growing trees by searching for best feature among a random subset of features
- This results a better model with higher bias and lower variance

Extremely Randomized Trees (Extra-Trees)

- Uses random thresholds for each feature rather than searching for best possible thresholds.
- Its makes trees even more random.
- Extra-Trees are much faster to train than regular Random Forests.
- One of Extra-Trees implementations is Scikit-Learn's ExtraTreeClassifier and ExtraTreeRegressor.
- Performance comparison is generally done over cross-validation to decide on one of these classifiers.

Feature Importance in Random Forests

- Makes it easy to measure relative importance of each feature.
- It is measured over how much tree nodes (across all trees) that uses a feature reduced impurity on average.
- It is a weighted average, where each node's weight is equal to the number of training samples that are associated with it.
- The normalized values (add up to 1) for importance of features are available through property feature_importances_.

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
...     print(name, score)
...
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Boosting

- An ensemble method that combines several weak learners into a strong learner.
- Predictors are trained sequentially while trying to correct its predecessor.
- Popular boosting methods are AdaBoost (Adaptive Boosting) and Gradient Boosting.

AdaBoost (Adaptive Boosting)

PLACE HOLDER

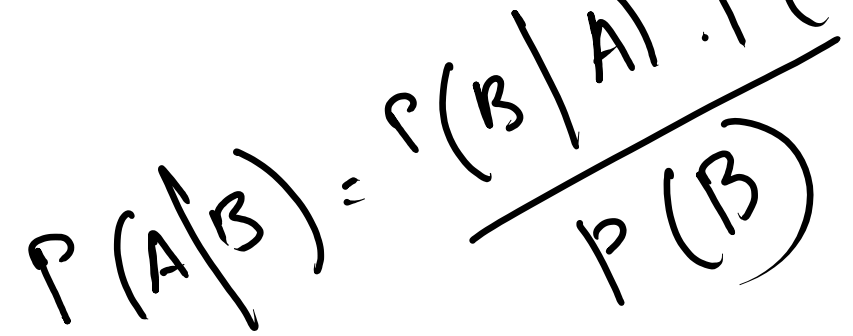
Bayesian Learning

A probabilistic approach for inference.

Naive Bayes Classifier

- One of the highly practical Bayesian learning methods.
- Performance is comparable to that of Neural Network and Decision Tree learning.
- Learning Task:
  - **Training instance *x***: Conjunction of attribute values.
  - **Target function *f(x)***: Any value from a finite set *V*.
  - **Training set**: A set of training examples of target function.
  - **Test instance**: Tuple of attribute values $<a_1, a_2, \ldots, a_n>$
  - **Task**: Prediction of target value for test instance

Applying Bayes Theorem



$$v_{NB} = argmax_{v_j \in V} P(v_j | a_1, a_2, \ldots, a_n)$$

v_{NB}: Classifier output; $P(v_j)$: frequency with which each target value v_j occurs in training data.

Applying Bayes theorem, the expression is rewritten as

$$v_{NB} = argmax_{v_j \in V} \frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, a_2, \ldots, a_n)}$$

$$= argmax_{v_j \in V} P(a_1, a_2, \ldots, a_n | v_j) P(v_j)$$

The term $P(a_1, a_2, \ldots, a_n)$ is dropped as it is a constant and independent of v_j.

Applying Bayes Theorem...Cont.

With the assumption that the attribute values are conditionally independent given the target value, the probability of observing conjunction a_1, a_2, ..., a_n is just the product of the probabilities for the individual attributes as shown in the expression below.

$$P(a_1, a_2, \ldots, a_n | v_j) = \prod_i P(a_i | v_i)$$

Now, substituting this into previous equation, we get

$$v_{NB} = argmax_{v_j \in V} (\prod_i P(a_i | v_i)) P(v_j)$$

$$v_{NB} = argmax_{v_j \in V} P(v_j) \prod_i P(a_i | v_i)$$

The Naïve Bayes Classifier

Example of Naïve Bayes Classification

- **Data Set:** PlayTennis
- **Test Instance:**

{

"Outlook": "Sunny",
"Temperature": "Cool",
"Humidity": "High",
"Wind": "Strong"

}

- **Task:** To classify the above-mentioned instance

Refer implementation at https://github.com/PradipKumarDas/Teaching/blob/master/Machine_Learning_Laboratory_Course_18AIL66/Program_%236/Naive_Bayes_Classification.ipynb

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

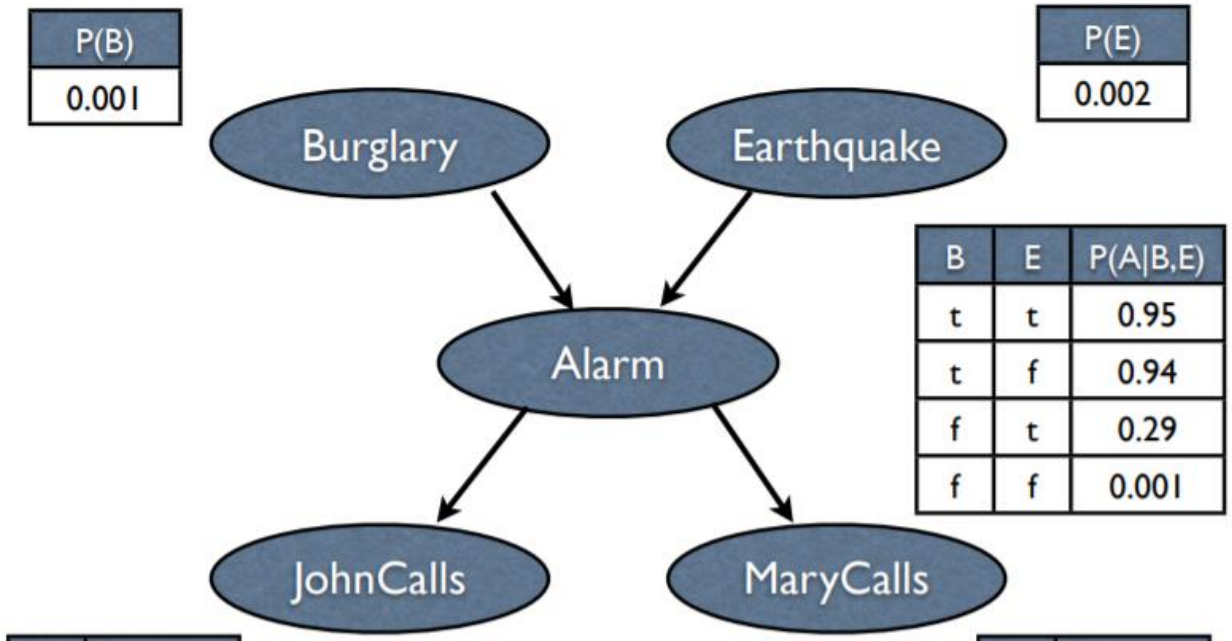
Example: Applying Naïve Bayes Classification into Text Classification

- **Instance Space X**: Consists of all possible text documents
- **Learning Task**:
  - **Training instance *x***: Conjunction of attribute values.
  - **Target function *f(x)***: Any value from a finite set *V* having value *like* and *dislike*.
  - **Training set**: A set of training examples of target function.
  - **Test instance**: Tuple of attribute values $<a_1, a_2, \ldots, a_n>$
  - **Task**: Prediction of target value for test instance
- **Design Aspect**:
  - Representing an arbitrary text document in terms of attribute values
  - Deciding estimating probabilities required by Naive Bayes Classifier

Bayesian (Belief) Networks

- Naive Bayes classifier assumes that all the attributes $a_1 ... a_n$ are conditionally independent given a target value v.
- But Bayesian belief network states conditional independence assumption that apply to subsets of the variables.
- Bayesian belief network describes probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities.
- In general, Bayesian belief network describes joint probability distribution over a set of variables.

A Typical Bayesian Network



Expectation-Maximization (EM) Algorithm

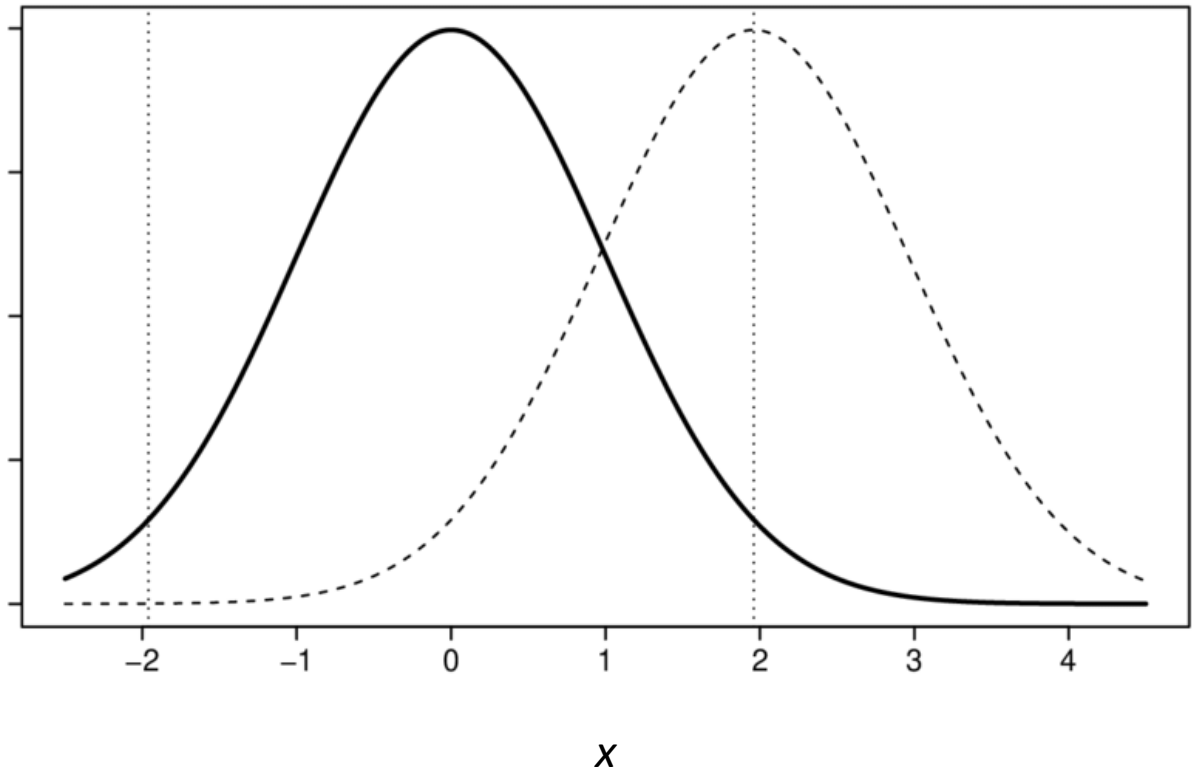
- It is a general approach to learn in presence of unobserved variables.
- It is used to train
  - Bayesian belief networks and
  - many unsupervised clustering algorithms.

Estimating Means of *K*-Normal Distributions

- The learning task is to output a hypothesis $h = \langle \mu_1, \ldots, \mu_k \rangle$ that describes the means of each of the *k* distributions.
- Each data instance is generated using a two-step process.
  - First, one of the *k* Normal distributions is selected at random.
  - Second, a single random instance x_i is generated according to this selected distribution.
- Assumptions:
  - Selection of single Normal distribution at each step is based on choosing each with uniform probability.
  - Each of the *k* Normal distributions has the same variance σ^2.

Example: *Two*-Normal Distributions

- Instances generated by a mixture of two Normal distributions with identical variance σ. The instances are shown by the points along the *x* axis.
- If the means of the Normal distributions are unknown, the EM algorithm can be used to search for their maximum likelihood estimates.
- Each instance is considered as triple (x_i, z_{i1} , z_{i2}), where x_i is the observed value of the i^{th} instance and z_{i1} and z_{i2} indicate which of the two Normal distributions was used to generate the value x_i.



Example: *Two*-Normal Distributions

The High-level Steps in EM Algorithm

- The algorithm first initializes the hypothesis to $h = \langle \mu_1, \mu_2 \rangle$), where μ_1 and μ_2 are arbitrary initial values.
- It then iteratively re-estimates h by repeating the following two steps (assuming the current hypothesis is correct) until the procedure converges to a stationary value for h.
  - **STEP 1:** Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij}, assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.
  - **STEP 2:** Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in STEP 1. Then replace the hypothesis $h = \langle \mu_1, \mu_2 \rangle$ by the new hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$ and iterate.

References

Textbooks:

1. Tom M. Mitchell. Machine Learning, McGraw-Hill Education, 2013
2. Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly, 2019

Annexture

Learning Tasks & Algorithms

Task	Algorithms	Models
Classification	Logistic Regression, Support Vector Machines (SVMs), K-Nearest Neighbours (KNN), Decision Trees, Random Forests, Neural Networks	Binary Classifiers: LogisticRegression, LinearSVC, SVC Multiclass Classifiers: Multinomial Logistic Regression, SGDClassifier, RandomForestClassifier, ExtraTreeClassifier, Naïve Bayes Multilabel Classifiers: MultinomialNB, KNeighborsClassifier
Regression	Linear Regression, Support Vector Machines (SVMs), Decision Trees, Random Forests, Neural Networks	Linear Regression, Regularized Linear Models (Ridge, Lasso & Elastic Net), LinearSVR, SVR(degree=), DecisionTreeRegressor, RandomForestRegressor, ExtraTreeRegressor
Clustering	K-Means	
Dimensionality Reduction	Principal Component Analysis (PCA)	
Visualization	t-Distributed Stochastic Neighbor Embedding (t-SNE)	
Anomaly and Novelty Detection		
Association Rule Learning		