Probabilistic Machine Learning

An Introduction

Kevin P. Murphy

Probabilistic Machine Learning

Adaptive Computation and Machine Learning
Thomas Dietterich, Editor
Christopher Bishop, David Heckerman, Michael Jordan, and Michael Kearns, Associate Editors

Bioinformatics: The Machine Learning Approach, Pierre Baldi and Søren Brunak
Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto
Graphical Models for Machine Learning and Digital Communication, Brendan J. Frey
Learning in Graphical Models, Michael I. Jordan
Causation, Prediction, and Search, second edition, Peter Spirtes, Clark Glymour, and Richard Scheines
Principles of Data Mining, David Hand, Heikki Mannila, and Padhraic Smyth
Bioinformatics: The Machine Learning Approach, second edition, Pierre Baldi and Søren Brunak
Learning Kernel Classifiers: Theory and Algorithms, Ralf Herbrich
Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, Bernhard Schölkopf and Alexander J. Smola
Introduction to Machine Learning, Ethem Alpaydin
Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K.I. Williams
Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, Eds.
The Minimum Description Length Principle, Peter D. Grünwald
Introduction to Statistical Relational Learning, Lise Getoor and Ben Taskar, Eds.
Probabilistic Graphical Models: Principles and Techniques, Daphne Koller and Nir Friedman
Introduction to Machine Learning, second edition, Ethem Alpaydin
Boosting: Foundations and Algorithms, Robert E. Schapire and Yoav Freund
Machine Learning: A Probabilistic Perspective, Kevin P. Murphy
Foundations of Machine Learning, Mehryar Mohri, Afshin Rostami, and Ameet Talwalker
Probabilistic Machine Learning: An Introduction, Kevin P. Murphy

Probabilistic Machine Learning

An Introduction

Kevin P. Murphy

The MIT Press
Cambridge, Massachusetts
London, England

© 2022 Massachusetts Institute of Technology

This work is subject to a Creative Commons CC-BY-NC-ND license.

Subject to such license, all rights are reserved.

The MIT Press would like to thank the anonymous peer reviewers who provided comments on drafts of this book. The generous work of academic experts is essential for establishing the authority and quality of our publications. We acknowledge with gratitude the contributions of these otherwise uncredited readers.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Murphy, Kevin P., author.
Title: Probabilistic machine learning : an introduction / Kevin P. Murphy.
Description: Cambridge, Massachusetts : The MIT Press, [2022]
Series: Adaptive computation and machine learning series
Includes bibliographical references and index.
Identifiers: LCCN 2021027430 | ISBN 9780262046824 (hardcover)
Subjects: LCSH: Machine learning. | Probabilities.
Classification: LCC Q325.5 .M872 2022 | DDC 006.3/1–dc23
LC record available at https://lccn.loc.gov/2021027430

10 9 8 7 6 5 4 3 2 1

This book is dedicated to my mother, Brigid Murphy,
who introduced me to the joy of learning and teaching.

Brief Contents

1 Introduction 1

I Foundations 31

2 Probability: Univariate Models 33
3 Probability: Multivariate Models 77
4 Statistics 107
5 Decision Theory 167
6 Information Theory 205
7 Linear Algebra 227
8 Optimization 273

II Linear Models 319

9 Linear Discriminant Analysis 321
10 Logistic Regression 337
11 Linear Regression 369
12 Generalized Linear Models * 413

III Deep Neural Networks 421

13 Neural Networks for Tabular Data 423
14 Neural Networks for Images 465
15 Neural Networks for Sequences 501

IV Nonparametric Models 543

16 Exemplar-based Methods 545
17 Kernel Methods * 565
18 Trees, Forests, Bagging, and Boosting 601

V Beyond Supervised Learning 623
19 Learning with Fewer Labeled Examples 625
20 Dimensionality Reduction 655
21 Clustering 713
22 Recommender Systems 739
23 Graph Embeddings * 751
A Notation 771

Contents

Preface **xxvii**

1 Introduction **1**

- 1.1 What is machine learning? 1
- 1.2 Supervised learning 1
  - 1.2.1 Classification 2
  - 1.2.2 Regression 8
  - 1.2.3 Overfitting and generalization 12
  - 1.2.4 No free lunch theorem 13
- 1.3 Unsupervised learning 14
  - 1.3.1 Clustering 14
  - 1.3.2 Discovering latent “factors of variation” 15
  - 1.3.3 Self-supervised learning 16
  - 1.3.4 Evaluating unsupervised learning 16
- 1.4 Reinforcement learning 17
- 1.5 Data 19
  - 1.5.1 Some common image datasets 19
  - 1.5.2 Some common text datasets 22
  - 1.5.3 Preprocessing discrete input data 23
  - 1.5.4 Preprocessing text data 24
  - 1.5.5 Handling missing data 27
- 1.6 Discussion 27
  - 1.6.1 The relationship between ML and other fields 27
  - 1.6.2 Structure of the book 28
  - 1.6.3 Caveats 28

I Foundations **31**

2 Probability: Univariate Models **33**

- 2.1 Introduction 33
  - 2.1.1 What is probability? 33

2.1.2 Types of uncertainty 33
2.1.3 Probability as an extension of logic 34
2.2 Random variables 35
2.2.1 Discrete random variables 35
2.2.2 Continuous random variables 36
2.2.3 Sets of related random variables 38
2.2.4 Independence and conditional independence 39
2.2.5 Moments of a distribution 40
2.2.6 Limitations of summary statistics * 43
2.3 Bayes’ rule 44
2.3.1 Example: Testing for COVID-19 46
2.3.2 Example: The Monty Hall problem 47
2.3.3 Inverse problems * 49
2.4 Bernoulli and binomial distributions 49
2.4.1 Definition 49
2.4.2 Sigmoid (logistic) function 50
2.4.3 Binary logistic regression 52
2.5 Categorical and multinomial distributions 53
2.5.1 Definition 53
2.5.2 Softmax function 54
2.5.3 Multiclass logistic regression 55
2.5.4 Log-sum-exp trick 56
2.6 Univariate Gaussian (normal) distribution 57
2.6.1 Cumulative distribution function 57
2.6.2 Probability density function 58
2.6.3 Regression 59
2.6.4 Why is the Gaussian distribution so widely used? 60
2.6.5 Dirac delta function as a limiting case 60
2.7 Some other common univariate distributions * 61
2.7.1 Student t distribution 61
2.7.2 Cauchy distribution 62
2.7.3 Laplace distribution 63
2.7.4 Beta distribution 63
2.7.5 Gamma distribution 64
2.7.6 Empirical distribution 65
2.8 Transformations of random variables * 66
2.8.1 Discrete case 66
2.8.2 Continuous case 66
2.8.3 Invertible transformations (bijections) 66
2.8.4 Moments of a linear transformation 69
2.8.5 The convolution theorem 70
2.8.6 Central limit theorem 71
2.8.7 Monte Carlo approximation 72
2.9 Exercises 73

3 Probability: Multivariate Models 77
- 3.1 Joint distributions for multiple random variables 77
  - 3.1.1 Covariance 77
  - 3.1.2 Correlation 78
  - 3.1.3 Uncorrelated does not imply independent 79
  - 3.1.4 Correlation does not imply causation 79
  - 3.1.5 Simpson's paradox 80
- 3.2 The multivariate Gaussian (normal) distribution 80
  - 3.2.1 Definition 81
  - 3.2.2 Mahalanobis distance 83
  - 3.2.3 Marginals and conditionals of an MVN * 84
  - 3.2.4 Example: conditioning a 2d Gaussian 85
  - 3.2.5 Example: Imputing missing values * 85
- 3.3 Linear Gaussian systems * 86
  - 3.3.1 Bayes rule for Gaussians 87
  - 3.3.2 Derivation * 87
  - 3.3.3 Example: Inferring an unknown scalar 88
  - 3.3.4 Example: inferring an unknown vector 90
  - 3.3.5 Example: sensor fusion 92
- 3.4 The exponential family * 93
  - 3.4.1 Definition 93
  - 3.4.2 Example 94
  - 3.4.3 Log partition function is cumulant generating function 95
  - 3.4.4 Maximum entropy derivation of the exponential family 95
- 3.5 Mixture models 96
  - 3.5.1 Gaussian mixture models 97
  - 3.5.2 Bernoulli mixture models 98
- 3.6 Probabilistic graphical models * 99
  - 3.6.1 Representation 100
  - 3.6.2 Inference 102
  - 3.6.3 Learning 102
- 3.7 Exercises 103

4 Statistics 107
- 4.1 Introduction 107
- 4.2 Maximum likelihood estimation (MLE) 107
  - 4.2.1 Definition 107
  - 4.2.2 Justification for MLE 108
  - 4.2.3 Example: MLE for the Bernoulli distribution 110
  - 4.2.4 Example: MLE for the categorical distribution 111
  - 4.2.5 Example: MLE for the univariate Gaussian 111
  - 4.2.6 Example: MLE for the multivariate Gaussian 112
  - 4.2.7 Example: MLE for linear regression 114
- 4.3 Empirical risk minimization (ERM) 115
  - 4.3.1 Example: minimizing the misclassification rate 116

- 4.3.2 Surrogate loss 116
- 4.4 Other estimation methods * 117
  - 4.4.1 The method of moments 117
  - 4.4.2 Online (recursive) estimation 119
- 4.5 Regularization 120
  - 4.5.1 Example: MAP estimation for the Bernoulli distribution 121
  - 4.5.2 Example: MAP estimation for the multivariate Gaussian * 122
  - 4.5.3 Example: weight decay 123
  - 4.5.4 Picking the regularizer using a validation set 124
  - 4.5.5 Cross-validation 125
  - 4.5.6 Early stopping 126
  - 4.5.7 Using more data 127
- 4.6 Bayesian statistics * 129
  - 4.6.1 Conjugate priors 129
  - 4.6.2 The beta-binomial model 130
  - 4.6.3 The Dirichlet-multinomial model 137
  - 4.6.4 The Gaussian-Gaussian model 141
  - 4.6.5 Beyond conjugate priors 144
  - 4.6.6 Credible intervals 146
  - 4.6.7 Bayesian machine learning 147
  - 4.6.8 Computational issues 151
- 4.7 Frequentist statistics * 154
  - 4.7.1 Sampling distributions 154
  - 4.7.2 Gaussian approximation of the sampling distribution of the MLE 155
  - 4.7.3 Bootstrap approximation of the sampling distribution of any estimator 156
  - 4.7.4 Confidence intervals 157
  - 4.7.5 Caution: Confidence intervals are not credible 158
  - 4.7.6 The bias-variance tradeoff 159
- 4.8 Exercises 164

5 Decision Theory 167

- 5.1 Bayesian decision theory 167
  - 5.1.1 Basics 167
  - 5.1.2 Classification problems 169
  - 5.1.3 ROC curves 171
  - 5.1.4 Precision-recall curves 174
  - 5.1.5 Regression problems 176
  - 5.1.6 Probabilistic prediction problems 177
- 5.2 Choosing the “right” model 179
  - 5.2.1 Bayesian hypothesis testing 179
  - 5.2.2 Bayesian model selection 180
  - 5.2.3 Occam’s razor 182
  - 5.2.4 Connection between cross validation and marginal likelihood 184
  - 5.2.5 Information criteria 185
  - 5.2.6 Posterior inference over effect sizes and Bayesian significance testing 186

5.3 Frequentist decision theory 188
  5.3.1 Computing the risk of an estimator 188
  5.3.2 Consistent estimators 191
  5.3.3 Admissible estimators 191
5.4 Empirical risk minimization 192
  5.4.1 Empirical risk 192
  5.4.2 Structural risk 194
  5.4.3 Cross-validation 194
  5.4.4 Statistical learning theory * 195
5.5 Frequentist hypothesis testing * 197
  5.5.1 Likelihood ratio test 197
  5.5.2 Type I vs type II errors and the Neyman-Pearson lemma 198
  5.5.3 Null hypothesis significance testing (NHST) and p-values 198
  5.5.4 p-values considered harmful 199
  5.5.5 Why isn't everyone a Bayesian? 201
5.6 Exercises 203

6 Information Theory 205

6.1 Entropy 205
  6.1.1 Entropy for discrete random variables 205
  6.1.2 Cross entropy 207
  6.1.3 Joint entropy 207
  6.1.4 Conditional entropy 208
  6.1.5 Perplexity 209
  6.1.6 Differential entropy for continuous random variables * 210
6.2 Relative entropy (KL divergence) * 211
  6.2.1 Definition 211
  6.2.2 Interpretation 212
  6.2.3 Example: KL divergence between two Gaussians 212
  6.2.4 Non-negativity of KL 212
  6.2.5 KL divergence and MLE 213
  6.2.6 Forward vs reverse KL 214
6.3 Mutual information * 215
  6.3.1 Definition 215
  6.3.2 Interpretation 216
  6.3.3 Example 216
  6.3.4 Conditional mutual information 217
  6.3.5 MI as a "generalized correlation coefficient" 218
  6.3.6 Normalized mutual information 219
  6.3.7 Maximal information coefficient 219
  6.3.8 Data processing inequality 221
  6.3.9 Sufficient Statistics 222
  6.3.10 Fano's inequality * 223
6.4 Exercises 224

7 Linear Algebra 227
- 7.1 Introduction 227
  - 7.1.1 Notation 227
  - 7.1.2 Vector spaces 230
  - 7.1.3 Norms of a vector and matrix 232
  - 7.1.4 Properties of a matrix 234
  - 7.1.5 Special types of matrices 237
- 7.2 Matrix multiplication 240
  - 7.2.1 Vector–vector products 240
  - 7.2.2 Matrix–vector products 241
  - 7.2.3 Matrix–matrix products 241
  - 7.2.4 Application: manipulating data matrices 243
  - 7.2.5 Kronecker products * 246
  - 7.2.6 Einstein summation * 246
- 7.3 Matrix inversion 247
  - 7.3.1 The inverse of a square matrix 247
  - 7.3.2 Schur complements * 248
  - 7.3.3 The matrix inversion lemma * 249
  - 7.3.4 Matrix determinant lemma * 249
  - 7.3.5 Application: deriving the conditionals of an MVN * 250
- 7.4 Eigenvalue decomposition (EVD) 251
  - 7.4.1 Basics 251
  - 7.4.2 Diagonalization 252
  - 7.4.3 Eigenvalues and eigenvectors of symmetric matrices 253
  - 7.4.4 Geometry of quadratic forms 254
  - 7.4.5 Standardizing and whitening data 254
  - 7.4.6 Power method 256
  - 7.4.7 Deflation 257
  - 7.4.8 Eigenvectors optimize quadratic forms 257
- 7.5 Singular value decomposition (SVD) 257
  - 7.5.1 Basics 257
  - 7.5.2 Connection between SVD and EVD 258
  - 7.5.3 Pseudo inverse 259
  - 7.5.4 SVD and the range and null space of a matrix * 260
  - 7.5.5 Truncated SVD 262
- 7.6 Other matrix decompositions * 262
  - 7.6.1 LU factorization 262
  - 7.6.2 QR decomposition 263
  - 7.6.3 Cholesky decomposition 264
- 7.7 Solving systems of linear equations * 264
  - 7.7.1 Solving square systems 265
  - 7.7.2 Solving underconstrained systems (least norm estimation) 265
  - 7.7.3 Solving overconstrained systems (least squares estimation) 266
- 7.8 Matrix calculus 267
  - 7.8.1 Derivatives 267

7.8.2 Gradients 268
7.8.3 Directional derivative 268
7.8.4 Total derivative * 269
7.8.5 Jacobian 269
7.8.6 Hessian 270
7.8.7 Gradients of commonly used functions 270
7.9 Exercises 272

8 Optimization 273
8.1 Introduction 273
8.1.1 Local vs global optimization 273
8.1.2 Constrained vs unconstrained optimization 275
8.1.3 Convex vs nonconvex optimization 275
8.1.4 Smooth vs nonsmooth optimization 279
8.2 First-order methods 280
8.2.1 Descent direction 282
8.2.2 Step size (learning rate) 282
8.2.3 Convergence rates 284
8.2.4 Momentum methods 285
8.3 Second-order methods 287
8.3.1 Newton's method 287
8.3.2 BFGS and other quasi-Newton methods 288
8.3.3 Trust region methods 289
8.4 Stochastic gradient descent 290
8.4.1 Application to finite sum problems 291
8.4.2 Example: SGD for fitting linear regression 291
8.4.3 Choosing the step size (learning rate) 292
8.4.4 Iterate averaging 295
8.4.5 Variance reduction * 295
8.4.6 Preconditioned SGD 296
8.5 Constrained optimization 299
8.5.1 Lagrange multipliers 300
8.5.2 The KKT conditions 302
8.5.3 Linear programming 303
8.5.4 Quadratic programming 304
8.5.5 Mixed integer linear programming * 305
8.6 Proximal gradient method * 306
8.6.1 Projected gradient descent 306
8.6.2 Proximal operator for ℓ_1-norm regularizer 308
8.6.3 Proximal operator for quantization 309
8.6.4 Incremental (online) proximal methods 309
8.7 Bound optimization * 310
8.7.1 The general algorithm 310
8.7.2 The EM algorithm 310
8.7.3 Example: EM for a GMM 313

8.8 Blackbox and derivative free optimization 317
8.9 Exercises 318

II Linear Models 319

9 Linear Discriminant Analysis 321
9.1 Introduction 321
9.2 Gaussian discriminant analysis 321
  9.2.1 Quadratic decision boundaries 322
  9.2.2 Linear decision boundaries 323
  9.2.3 The connection between LDA and logistic regression 323
  9.2.4 Model fitting 324
  9.2.5 Nearest centroid classifier 326
  9.2.6 Fisher's linear discriminant analysis * 326
9.3 Naive Bayes classifiers 330
  9.3.1 Example models 330
  9.3.2 Model fitting 331
  9.3.3 Bayesian naive Bayes 332
  9.3.4 The connection between naive Bayes and logistic regression 333
9.4 Generative vs discriminative classifiers 334
  9.4.1 Advantages of discriminative classifiers 334
  9.4.2 Advantages of generative classifiers 335
  9.4.3 Handling missing features 335
9.5 Exercises 336

10 Logistic Regression 337
10.1 Introduction 337
10.2 Binary logistic regression 337
  10.2.1 Linear classifiers 337
  10.2.2 Nonlinear classifiers 338
  10.2.3 Maximum likelihood estimation 340
  10.2.4 Stochastic gradient descent 343
  10.2.5 Perceptron algorithm 344
  10.2.6 Iteratively reweighted least squares 344
  10.2.7 MAP estimation 346
  10.2.8 Standardization 348
10.3 Multinomial logistic regression 348
  10.3.1 Linear and nonlinear classifiers 349
  10.3.2 Maximum likelihood estimation 349
  10.3.3 Gradient-based optimization 352
  10.3.4 Bound optimization 352
  10.3.5 MAP estimation 353
  10.3.6 Maximum entropy classifiers 354
  10.3.7 Hierarchical classification 355

- 10.3.8 Handling large numbers of classes 356
- 10.4 Robust logistic regression * 358
  - 10.4.1 Mixture model for the likelihood 358
  - 10.4.2 Bi-tempered loss 359
- 10.5 Bayesian logistic regression * 361
  - 10.5.1 Laplace approximation 361
  - 10.5.2 Approximating the posterior predictive 364
- 10.6 Exercises 365

11 Linear Regression 369

- 11.1 Introduction 369
- 11.2 Least squares linear regression 369
  - 11.2.1 Terminology 369
  - 11.2.2 Least squares estimation 370
  - 11.2.3 Other approaches to computing the MLE 374
  - 11.2.4 Measuring goodness of fit 378
- 11.3 Ridge regression 379
  - 11.3.1 Computing the MAP estimate 380
  - 11.3.2 Connection between ridge regression and PCA 381
  - 11.3.3 Choosing the strength of the regularizer 382
- 11.4 Lasso regression 383
  - 11.4.1 MAP estimation with a Laplace prior (ℓ_1 regularization) 383
  - 11.4.2 Why does ℓ_1 regularization yield sparse solutions? 384
  - 11.4.3 Hard vs soft thresholding 385
  - 11.4.4 Regularization path 387
  - 11.4.5 Comparison of least squares, lasso, ridge and subset selection 388
  - 11.4.6 Variable selection consistency 390
  - 11.4.7 Group lasso 391
  - 11.4.8 Elastic net (ridge and lasso combined) 394
  - 11.4.9 Optimization algorithms 395
- 11.5 Regression splines * 397
  - 11.5.1 B-spline basis functions 397
  - 11.5.2 Fitting a linear model using a spline basis 399
  - 11.5.3 Smoothing splines 399
  - 11.5.4 Generalized additive models 399
- 11.6 Robust linear regression * 400
  - 11.6.1 Laplace likelihood 400
  - 11.6.2 Student-t likelihood 402
  - 11.6.3 Huber loss 402
  - 11.6.4 RANSAC 402
- 11.7 Bayesian linear regression * 403
  - 11.7.1 Priors 403
  - 11.7.2 Posteriors 403
  - 11.7.3 Example 404
  - 11.7.4 Computing the posterior predictive 404

    11.7.5 The advantage of centering 406
    11.7.6 Dealing with multicollinearity 407
    11.7.7 Automatic relevancy determination (ARD) * 408
  11.8 Exercises 409

12 Generalized Linear Models * 413
  12.1 Introduction 413
  12.2 Examples 413
    12.2.1 Linear regression 414
    12.2.2 Binomial regression 414
    12.2.3 Poisson regression 415
  12.3 GLMs with non-canonical link functions 415
  12.4 Maximum likelihood estimation 416
  12.5 Worked example: predicting insurance claims 417

III Deep Neural Networks 421

13 Neural Networks for Tabular Data 423
  13.1 Introduction 423
  13.2 Multilayer perceptrons (MLPs) 424
    13.2.1 The XOR problem 425
    13.2.2 Differentiable MLPs 426
    13.2.3 Activation functions 426
    13.2.4 Example models 428
    13.2.5 The importance of depth 432
    13.2.6 The “deep learning revolution” 433
    13.2.7 Connections with biology 434
  13.3 Backpropagation 436
    13.3.1 Forward vs reverse mode differentiation 436
    13.3.2 Reverse mode differentiation for multilayer perceptrons 438
    13.3.3 Vector-Jacobian product for common layers 439
    13.3.4 Computation graphs 442
  13.4 Training neural networks 444
    13.4.1 Tuning the learning rate 445
    13.4.2 Vanishing and exploding gradients 445
    13.4.3 Non-saturating activation functions 446
    13.4.4 Residual connections 449
    13.4.5 Parameter initialization 450
    13.4.6 Parallel training 452
  13.5 Regularization 453
    13.5.1 Early stopping 453
    13.5.2 Weight decay 453
    13.5.3 Sparse DNNs 453
    13.5.4 Dropout 453

- 13.5.5 Bayesian neural networks 455
- 13.5.6 Regularization effects of (stochastic) gradient descent * 455
- 13.6 Other kinds of feedforward networks * 457
  - 13.6.1 Radial basis function networks 457
  - 13.6.2 Mixtures of experts 459
- 13.7 Exercises 462

14 Neural Networks for Images 465

- 14.1 Introduction 465
- 14.2 Common layers 466
  - 14.2.1 Convolutional layers 466
  - 14.2.2 Pooling layers 473
  - 14.2.3 Putting it all together 474
  - 14.2.4 Normalization layers 474
- 14.3 Common architectures for image classification 477
  - 14.3.1 LeNet 477
  - 14.3.2 AlexNet 479
  - 14.3.3 GoogLeNet (Inception) 480
  - 14.3.4 ResNet 481
  - 14.3.5 DenseNet 482
  - 14.3.6 Neural architecture search 483
- 14.4 Other forms of convolution * 484
  - 14.4.1 Dilated convolution 484
  - 14.4.2 Transposed convolution 484
  - 14.4.3 Depthwise separable convolution 486
- 14.5 Solving other discriminative vision tasks with CNNs * 486
  - 14.5.1 Image tagging 486
  - 14.5.2 Object detection 487
  - 14.5.3 Instance segmentation 488
  - 14.5.4 Semantic segmentation 489
  - 14.5.5 Human pose estimation 490
- 14.6 Generating images by inverting CNNs * 491
  - 14.6.1 Converting a trained classifier into a generative model 491
  - 14.6.2 Image priors 492
  - 14.6.3 Visualizing the features learned by a CNN 493
  - 14.6.4 Deep Dream 494
  - 14.6.5 Neural style transfer 495

15 Neural Networks for Sequences 501

- 15.1 Introduction 501
- 15.2 Recurrent neural networks (RNNs) 501
  - 15.2.1 Vec2Seq (sequence generation) 501
  - 15.2.2 Seq2Vec (sequence classification) 503
  - 15.2.3 Seq2Seq (sequence translation) 505
  - 15.2.4 Teacher forcing 507

15.2.5 Backpropagation through time 508
15.2.6 Vanishing and exploding gradients 509
15.2.7 Gating and long term memory 510
15.2.8 Beam search 513
15.3 1d CNNs 514
15.3.1 1d CNNs for sequence classification 514
15.3.2 Causal 1d CNNs for sequence generation 515
15.4 Attention 516
15.4.1 Attention as soft dictionary lookup 517
15.4.2 Kernel regression as non-parametric attention 518
15.4.3 Parametric attention 519
15.4.4 Seq2Seq with attention 520
15.4.5 Seq2vec with attention (text classification) 521
15.4.6 Seq+Seq2Vec with attention (text pair classification) 521
15.4.7 Soft vs hard attention 523
15.5 Transformers 524
15.5.1 Self-attention 524
15.5.2 Multi-headed attention 525
15.5.3 Positional encoding 526
15.5.4 Putting it all together 527
15.5.5 Comparing transformers, CNNs and RNNs 529
15.5.6 Transformers for images * 530
15.5.7 Other transformer variants * 531
15.6 Efficient transformers * 531
15.6.1 Fixed non-learnable localized attention patterns 532
15.6.2 Learnable sparse attention patterns 533
15.6.3 Memory and recurrence methods 533
15.6.4 Low-rank and kernel methods 533
15.7 Language models and unsupervised representation learning 535
15.7.1 ELMo 536
15.7.2 BERT 536
15.7.3 GPT 540
15.7.4 T5 541
15.7.5 Discussion 541

IV Nonparametric Models 543

16 Exemplar-based Methods 545
16.1 K nearest neighbor (KNN) classification 545
16.1.1 Example 546
16.1.2 The curse of dimensionality 546
16.1.3 Reducing the speed and memory requirements 548
16.1.4 Open set recognition 548
16.2 Learning distance metrics 549

- 16.2.1 Linear and convex methods 550
- 16.2.2 Deep metric learning 552
- 16.2.3 Classification losses 552
- 16.2.4 Ranking losses 553
- 16.2.5 Speeding up ranking loss optimization 554
- 16.2.6 Other training tricks for DML 557
- 16.3 Kernel density estimation (KDE) 558
  - 16.3.1 Density kernels 558
  - 16.3.2 Parzen window density estimator 559
  - 16.3.3 How to choose the bandwidth parameter 560
  - 16.3.4 From KDE to KNN classification 561
  - 16.3.5 Kernel regression 561

17 Kernel Methods * 565

- 17.1 Mercer kernels 565
  - 17.1.1 Mercer's theorem 566
  - 17.1.2 Some popular Mercer kernels 567
- 17.2 Gaussian processes 572
  - 17.2.1 Noise-free observations 572
  - 17.2.2 Noisy observations 573
  - 17.2.3 Comparison to kernel regression 574
  - 17.2.4 Weight space vs function space 575
  - 17.2.5 Numerical issues 575
  - 17.2.6 Estimating the kernel 576
  - 17.2.7 GPs for classification 579
  - 17.2.8 Connections with deep learning 580
  - 17.2.9 Scaling GPs to large datasets 580
- 17.3 Support vector machines (SVMs) 583
  - 17.3.1 Large margin classifiers 583
  - 17.3.2 The dual problem 585
  - 17.3.3 Soft margin classifiers 587
  - 17.3.4 The kernel trick 588
  - 17.3.5 Converting SVM outputs into probabilities 589
  - 17.3.6 Connection with logistic regression 589
  - 17.3.7 Multi-class classification with SVMs 590
  - 17.3.8 How to choose the regularizer C 591
  - 17.3.9 Kernel ridge regression 592
  - 17.3.10 SVMs for regression 593
- 17.4 Sparse vector machines 595
  - 17.4.1 Relevance vector machines (RVMs) 596
  - 17.4.2 Comparison of sparse and dense kernel methods 596
- 17.5 Exercises 599

18 Trees, Forests, Bagging, and Boosting 601

- 18.1 Classification and regression trees (CART) 601

18.1.1 Model definition 601
18.1.2 Model fitting 603
18.1.3 Regularization 604
18.1.4 Handling missing input features 604
18.1.5 Pros and cons 604
18.2 Ensemble learning 606
18.2.1 Stacking 606
18.2.2 Ensembling is not Bayes model averaging 607
18.3 Bagging 607
18.4 Random forests 608
18.5 Boosting 609
18.5.1 Forward stagewise additive modeling 610
18.5.2 Quadratic loss and least squares boosting 610
18.5.3 Exponential loss and AdaBoost 611
18.5.4 LogitBoost 614
18.5.5 Gradient boosting 614
18.6 Interpreting tree ensembles 618
18.6.1 Feature importance 619
18.6.2 Partial dependency plots 621

V Beyond Supervised Learning 623

19 Learning with Fewer Labeled Examples 625
19.1 Data augmentation 625
19.1.1 Examples 625
19.1.2 Theoretical justification 626
19.2 Transfer learning 626
19.2.1 Fine-tuning 627
19.2.2 Adapters 628
19.2.3 Supervised pre-training 629
19.2.4 Unsupervised pre-training (self-supervised learning) 630
19.2.5 Domain adaptation 635
19.3 Semi-supervised learning 636
19.3.1 Self-training and pseudo-labeling 636
19.3.2 Entropy minimization 637
19.3.3 Co-training 640
19.3.4 Label propagation on graphs 641
19.3.5 Consistency regularization 642
19.3.6 Deep generative models * 644
19.3.7 Combining self-supervised and semi-supervised learning 647
19.4 Active learning 648
19.4.1 Decision-theoretic approach 648
19.4.2 Information-theoretic approach 648
19.4.3 Batch active learning 649

- 19.5 Meta-learning 649
  - 19.5.1 Model-agnostic meta-learning (MAML) 650
- 19.6 Few-shot learning 651
  - 19.6.1 Matching networks 651
- 19.7 Weakly supervised learning 653
- 19.8 Exercises 653

20 Dimensionality Reduction 655

- 20.1 Principal components analysis (PCA) 655
  - 20.1.1 Examples 655
  - 20.1.2 Derivation of the algorithm 657
  - 20.1.3 Computational issues 660
  - 20.1.4 Choosing the number of latent dimensions 662
- 20.2 Factor analysis * 664
  - 20.2.1 Generative model 665
  - 20.2.2 Probabilistic PCA 666
  - 20.2.3 EM algorithm for FA/PPCA 667
  - 20.2.4 Unidentifiability of the parameters 669
  - 20.2.5 Nonlinear factor analysis 671
  - 20.2.6 Mixtures of factor analysers 672
  - 20.2.7 Exponential family factor analysis 673
  - 20.2.8 Factor analysis models for paired data 675
- 20.3 Autoencoders 677
  - 20.3.1 Bottleneck autoencoders 678
  - 20.3.2 Denoising autoencoders 679
  - 20.3.3 Contractive autoencoders 680
  - 20.3.4 Sparse autoencoders 681
  - 20.3.5 Variational autoencoders 681
- 20.4 Manifold learning * 687
  - 20.4.1 What are manifolds? 687
  - 20.4.2 The manifold hypothesis 687
  - 20.4.3 Approaches to manifold learning 688
  - 20.4.4 Multi-dimensional scaling (MDS) 689
  - 20.4.5 Isomap 692
  - 20.4.6 Kernel PCA 692
  - 20.4.7 Maximum variance unfolding (MVU) 694
  - 20.4.8 Local linear embedding (LLE) 695
  - 20.4.9 Laplacian eigenmaps 696
  - 20.4.10 t-SNE 699
- 20.5 Word embeddings 703
  - 20.5.1 Latent semantic analysis / indexing 703
  - 20.5.2 Word2vec 705
  - 20.5.3 GloVE 707
  - 20.5.4 Word analogies 708
  - 20.5.5 RAND-WALK model of word embeddings 709

20.5.6 Contextual word embeddings 710
20.6 Exercises 710

21 Clustering 713

21.1 Introduction 713
21.1.1 Evaluating the output of clustering methods 713
21.2 Hierarchical agglomerative clustering 715
21.2.1 The algorithm 716
21.2.2 Example 718
21.2.3 Extensions 719
21.3 K means clustering 720
21.3.1 The algorithm 720
21.3.2 Examples 720
21.3.3 Vector quantization 722
21.3.4 The K-means++ algorithm 723
21.3.5 The K-medoids algorithm 723
21.3.6 Speedup tricks 724
21.3.7 Choosing the number of clusters K 724
21.4 Clustering using mixture models 727
21.4.1 Mixtures of Gaussians 728
21.4.2 Mixtures of Bernoullis 731
21.5 Spectral clustering * 732
21.5.1 Normalized cuts 732
21.5.2 Eigenvectors of the graph Laplacian encode the clustering 733
21.5.3 Example 734
21.5.4 Connection with other methods 735
21.6 Biclustering * 735
21.6.1 Basic biclustering 736
21.6.2 Nested partition models (Crosscat) 736

22 Recommender Systems 739

22.1 Explicit feedback 739
22.1.1 Datasets 739
22.1.2 Collaborative filtering 740
22.1.3 Matrix factorization 741
22.1.4 Autoencoders 743
22.2 Implicit feedback 745
22.2.1 Bayesian personalized ranking 745
22.2.2 Factorization machines 746
22.2.3 Neural matrix factorization 747
22.3 Leveraging side information 747
22.4 Exploration-exploitation tradeoff 748

23 Graph Embeddings * 751

23.1 Introduction 751

- 23.2 Graph Embedding as an Encoder/Decoder Problem 752
- 23.3 Shallow graph embeddings 754
  - 23.3.1 Unsupervised embeddings 755
  - 23.3.2 Distance-based: Euclidean methods 755
  - 23.3.3 Distance-based: non-Euclidean methods 756
  - 23.3.4 Outer product-based: Matrix factorization methods 756
  - 23.3.5 Outer product-based: Skip-gram methods 757
  - 23.3.6 Supervised embeddings 759
- 23.4 Graph Neural Networks 760
  - 23.4.1 Message passing GNNs 760
  - 23.4.2 Spectral Graph Convolutions 761
  - 23.4.3 Spatial Graph Convolutions 761
  - 23.4.4 Non-Euclidean Graph Convolutions 763
- 23.5 Deep graph embeddings 763
  - 23.5.1 Unsupervised embeddings 764
  - 23.5.2 Semi-supervised embeddings 766
- 23.6 Applications 767
  - 23.6.1 Unsupervised applications 767
  - 23.6.2 Supervised applications 769

A Notation 771

- A.1 Introduction 771
- A.2 Common mathematical symbols 771
- A.3 Functions 772
  - A.3.1 Common functions of one argument 772
  - A.3.2 Common functions of two arguments 772
  - A.3.3 Common functions of > 2 arguments 772
- A.4 Linear algebra 773
  - A.4.1 General notation 773
  - A.4.2 Vectors 773
  - A.4.3 Matrices 773
  - A.4.4 Matrix calculus 774
- A.5 Optimization 774
- A.6 Probability 775
- A.7 Information theory 775
- A.8 Statistics and machine learning 776
  - A.8.1 Supervised learning 776
  - A.8.2 Unsupervised learning and generative models 776
  - A.8.3 Bayesian inference 776
- A.9 Abbreviations 777

Index 779

Bibliography 796

Preface

In 2012, I published a 1200-page book called *Machine Learning: A Probabilistic Perspective*, which provided a fairly comprehensive coverage of the field of machine learning (ML) at that time, under the unifying lens of probabilistic modeling. The book was well received, and won the De Groot prize in 2013.

The year 2012 is also generally considered the start of the “deep learning revolution”. The term “deep learning” refers to a branch of ML that is based on neural networks (DNNs), which are nonlinear functions with many layers of processing (hence the term “deep”). Although this basic technology had been around for many years, it was in 2012 when [KSH12] used DNNs to win the ImageNet image classification challenge by such a large margin that it caught the attention of the wider community. Related advances on other hard problems, such as speech recognition, appeared around the same time (see e.g., [Cir+10; Cir+11; Hin+12]). These breakthroughs were enabled by advances in hardware technology (in particular, the repurposing of fast graphics processing units (GPUs) from video games to ML), data collection technology (in particular, the use of crowd sourcing tools, such as Amazon’s Mechanical Turk platform, to collect large labeled datasets, such as ImageNet), as well as various new algorithmic ideas, some of which we cover in this book.

Since 2012, the field of deep learning has exploded, with new advances coming at an increasing pace. Interest in the field has also grown rapidly, fueled by the commercial success of the technology, and the breadth of applications to which it can be applied. Therefore, in 2018, I decided to write a second edition of my book, to attempt to summarize some of this progress.

By March 2020, my draft of the second edition had swollen to about 1600 pages, and I still had many topics left to cover. As a result, MIT Press told me I would need to split the book into two volumes. Then the COVID-19 pandemic struck. I decided to pivot away from book writing, and to help develop the risk score algorithm for Google’s exposure notification app [MKS21] as well as to assist with various forecasting projects [Wah+22]. However, by the Fall of 2020, I decided to return to working on the book.

To make up for lost time, I asked several colleagues to help me finish by writing various sections (see acknowledgements below). The result of all this is two new books, “Probabilistic Machine Learning: An Introduction”, which you are currently reading, and “Probabilistic Machine Learning: Advanced Topics”, which is the sequel to this book [Mur23]. Together these two books attempt to present a fairly broad coverage of the field of ML c. 2021, using the same unifying lens of probabilistic modeling and Bayesian decision theory that I used in the 2012 book.

Nearly all of the content from the 2012 book has been retained, but it is now split fairly evenly between the two new books. In addition, each new book has lots of fresh material, covering topics from deep learning, as well as advances in other parts of the field, such as generative models, variational inference and reinforcement learning.

To make this introductory book more self-contained and useful for students, I have added some background material, on topics such as optimization and linear algebra, that was omitted from the 2012 book due to lack of space. Advanced material, that can be skipped during an introductory level course, is denoted by an asterisk * in the section or chapter title. Exercises can be found at the end of some chapters. Solutions to exercises marked with an asterisk * are available to qualified instructors by contacting MIT Press; solutions to all other exercises can be found online at `probml.github.io/book1`, along with additional teaching material (e.g., figures and slides).

Another major change is that all of the software now uses Python instead of Matlab. (In the future, we may create a Julia version of the code.) The new code leverages standard Python libraries, such as NumPy, Scikit-learn, JAX, PyTorch, TensorFlow, PyMC, etc.

If a figure caption says "Generated by `iris_plot.ipynb`", then you can find the corresponding Jupyter notebook at probml.github.io/notebooks#iris_plot.ipynb. Clicking on the figure link in the pdf version of the book will take you to this list of notebooks. Clicking on the notebook link will open it inside Google Colab, which will let you easily reproduce the figure for yourself, and modify the underlying source code to gain a deeper understanding of the methods. (Colab gives you access to a free GPU, which is useful for some of the more computationally heavy demos.)

Acknowledgements

I would like to thank the following people for helping me with the book:

- Zico Kolter (CMU), who helped write parts of Chapter 7 (Linear Algebra).
- Frederik Kunstner, Si Yi Meng, Aaron Mishkin, Sharan Vaswani, and Mark Schmidt who helped write parts of Chapter 8 (Optimization).
- Mathieu Blondel (Google), who helped write Section 13.3 (Backpropagation).
- Krzysztof Choromanski (Google), who wrote Section 15.6 (Efficient transformers *).
- Colin Raffel (UNC), who helped write Section 19.2 (Transfer learning) and Section 19.3 (Semi-supervised learning).
- Bryan Perozzi (Google), Sami Abu-El-Haija (USC) and Ines Chami, who helped write Chapter 23 (Graph Embeddings *).
- John Fearns and Peter Cerno for carefully proofreading the book.
- Many members of the github community for finding typos, etc (see `https://github.com/probml/pml-book/issues?q=is:issue` for a list of issues).
- The 4 anonymous reviewers solicited by MIT Press.
- Mahmoud Soliman for writing all the magic plumbing code that connects latex, colab, github, etc, and for teaching me about GCP and TPUs.
- The 2021 cohort of Google Summer of Code students who worked on code for the book: Aleyna Kara, Srikar Jilugu, Drishti Patel, Ming Liang Ang, Gerardo Durán-Martín. (See `https://probml.github.io/pml-book/gsoc/gsoc2021.html` for a summary of their contributions.)
- Zeel B Patel, Karm Patel, Nitish Sharma, Ankita Kumari Jain and Nipun Batra for help improving the figures and code after the book first came out.
- Many members of the github community for their code contributions (see `https://github.com/`

`probml/pyprobml#acknowledgements`).
- The authors of [Zha+20], [Gér17] and [Mar18] for letting me reuse or modify some of their open source code from their own excellent books.
- My manager at Google, Doug Eck, for letting me spend company time on this book.
- My wife Margaret for letting me spend family time on this book.

About the cover

The cover illustrates a neural network (Chapter 13) being used to classify a hand-written digit $\boldsymbol{x}$ into one of 10 class labels $y \in \{0, 1, \ldots, 9\}$. The histogram on the right is the output of the model, and corresponds to the conditional probability distribution $p(y|\boldsymbol{x})$.[1]

Changelog

Changes listed at `https://github.com/probml/pml-book/issues?q=is%3Aissue+is%3Aclosed`.

- August 8, 2022. First hard-copy printing.
- April 4, 2023. Second hard-copy printing.
- June 22, 2023: This online version.

---

[1] There is an error in the illustration; it accidently has 11 bins.

1 Introduction

1.1 What is machine learning?

A popular definition of **machine learning** or **ML**, due to Tom Mitchell [Mit97], is as follows:

> A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Thus there are many different kinds of machine learning, depending on the nature of the tasks T we wish the system to learn, the nature of the performance measure P we use to evaluate the system, and the nature of the training signal or experience E we give it.

In this book, we will cover the most common types of ML, but from a **probabilistic perspective**. Roughly speaking, this means that we treat all unknown quantities (e.g., predictions about the future value of some quantity of interest, such as tomorrow's temperature, or the parameters of some model) as **random variables**, that are endowed with **probability distributions** which describe a weighted set of possible values the variable may have. (See Chapter 2 for a quick refresher on the basics of probability, if necessary.)

There are two main reasons we adopt a probabilistic approach. First, it is the optimal approach to **decision making under uncertainty**, as we explain in Section 5.1. Second, probabilistic modeling is the language used by most other areas of science and engineering, and thus provides a unifying framework between these fields. As Shakir Mohamed, a researcher at DeepMind, put it:[1]

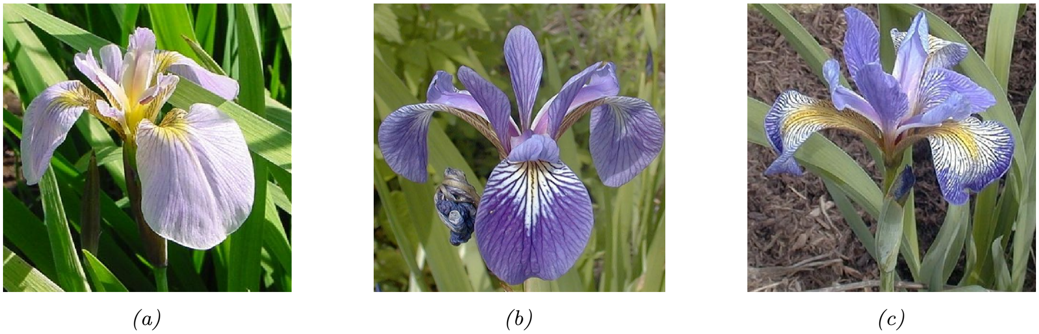
> Almost all of machine learning can be viewed in probabilistic terms, making probabilistic thinking fundamental. It is, of course, not the only view. But it is through this view that we can connect what we do in machine learning to every other computational science, whether that be in stochastic optimisation, control theory, operations research, econometrics, information theory, statistical physics or bio-statistics. For this reason alone, mastery of probabilistic thinking is essential.

1.2 Supervised learning

The most common form of ML is **supervised learning**. In this problem, the task T is to learn a mapping f from inputs $\boldsymbol{x} \in \mathcal{X}$ to outputs $\boldsymbol{y} \in \mathcal{Y}$. The inputs $\boldsymbol{x}$ are also called the **features**,

---

[1] Source: Slide 2 of `https://bit.ly/3pyHyPn`



(a) (b) (c)

Figure 1.1: Three types of Iris flowers: Setosa, Versicolor and Virginica. Used with kind permission of Dennis Kramb and SIGNA.

index	sl	sw	pl	pw	label
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
	...				
50	7.0	3.2	4.7	1.4	Versicolor
	...				
149	5.9	3.0	5.1	1.8	Virginica

Table 1.1: A subset of the Iris design matrix. The features are: sepal length, sepal width, petal length, petal width. There are 50 examples of each class.

covariates, or **predictors**; this is often a fixed-dimensional vector of numbers, such as the height and weight of a person, or the pixels in an image. In this case, $\mathcal{X} = \mathbb{R}^D$, where D is the dimensionality of the vector (i.e., the number of input features). The output $\boldsymbol{y}$ is also known as the **label**, **target**, or **response**.[2] The experience E is given in the form of a set of N input-output pairs $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$, known as the **training set**. (N is called the **sample size**.) The performance measure P depends on the type of output we are predicting, as we discuss below.

1.2.1 Classification

In **classification** problems, the output space is a set of C unordered and mutually exclusive labels known as **classes**, $\mathcal{Y} = \{1, 2, \ldots, C\}$. The problem of predicting the class label given an input is also called **pattern recognition**. (If there are just two classes, often denoted by $y \in \{0, 1\}$ or $y \in \{-1, +1\}$, it is called **binary classification**.)

1.2.1.1 Example: classifying Iris flowers

As an example, consider the problem of classifying Iris flowers into their 3 subspecies, Setosa, Versicolor and Virginica. Figure 1.1 shows one example of each of these classes.

---

[2] Sometimes (e.g., in the statsmodels Python package) $\boldsymbol{x}$ are called the **exogenous variables** and $\boldsymbol{y}$ are called the **endogenous variables**.

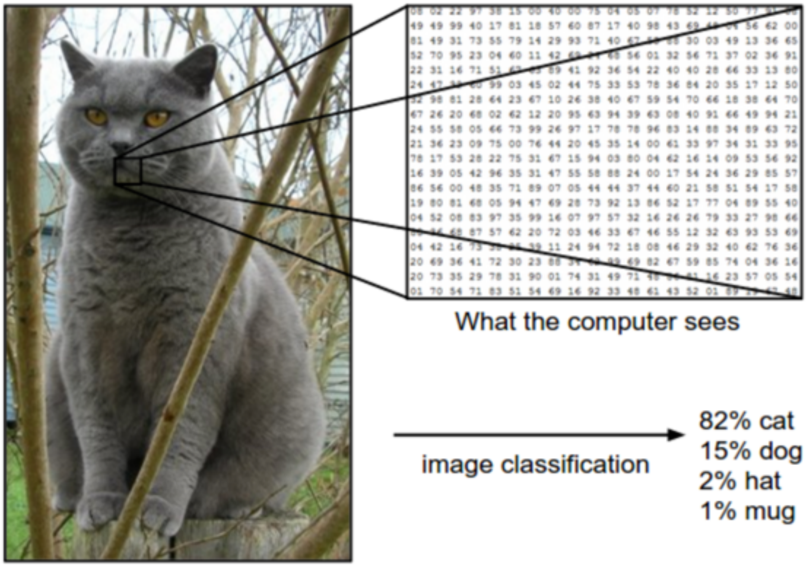


Figure 1.2: Illustration of the image classification problem. From https://cs231n.github.io/. Used with kind permission of Andrej Karpathy.

In **image classification**, the input space $\mathcal{X}$ is the set of images, which is a very high-dimensional space: for a color image with $C = 3$ channels (e.g., RGB) and $D_1 \times D_2$ pixels, we have $\mathcal{X} = \mathbb{R}^D$, where $D = C \times D_1 \times D_2$. (In practice we represent each pixel intensity with an integer, typically from the range $\{0, 1, \ldots, 255\}$, but we assume real valued inputs for notational simplicity.) Learning a mapping $f : \mathcal{X} \to \mathcal{Y}$ from images to labels is quite challenging, as illustrated in Figure 1.2. However, it can be tackled using certain kinds of functions, such as a **convolutional neural network** or **CNN**, which we discuss in Section 14.1.

Fortunately for us, some botanists have already identified 4 simple, but highly informative, numeric features — sepal length, sepal width, petal length, petal width — which can be used to distinguish the three kinds of Iris flowers. In this section, we will use this much lower-dimensional input space, $\mathcal{X} = \mathbb{R}^4$, for simplicity. The **Iris dataset** is a collection of 150 labeled examples of Iris flowers, 50 of each type, described by these 4 features. It is widely used as an example, because it is small and simple to understand. (We will discuss larger and more complex datasets later in the book.)

When we have small datasets of features, it is common to store them in an $N \times D$ matrix, in which each row represents an example, and each column represents a feature. This is known as a **design matrix**; see Table 1.1 for an example.[3]

The Iris dataset is an example of **tabular data**. When the inputs are of variable size (e.g., sequences of words, or social networks), rather than fixed-length vectors, the data is usually stored

---

[3] This particular design matrix has $N = 150$ rows and $D = 4$ columns, and hence has a **tall and skinny** shape, since $N \gg D$. By contrast, some datasets (e.g., genomics) have more features than examples, $D \gg N$; their design matrices are **short and fat**. The term "**big data**" usually means that N is large, whereas the term "**wide data**" means that D is large (relative to N).

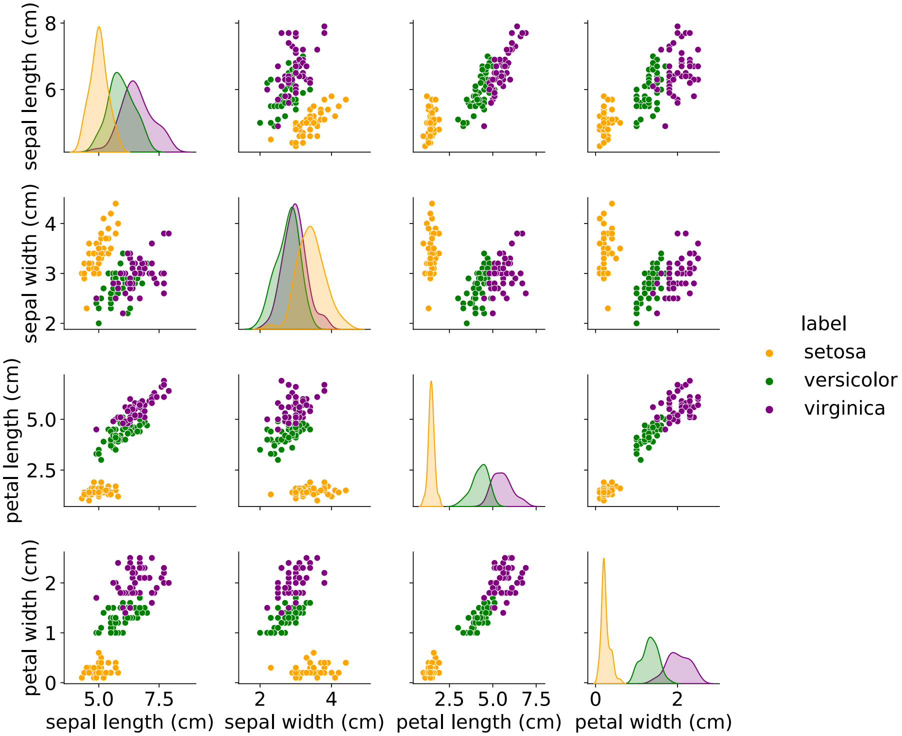


Figure 1.3: Visualization of the Iris data as a pairwise scatter plot. On the diagonal we plot the marginal distribution of each feature for each class. The off-diagonals contain scatterplots of all possible pairs of features. Generated by iris_plot.ipynb

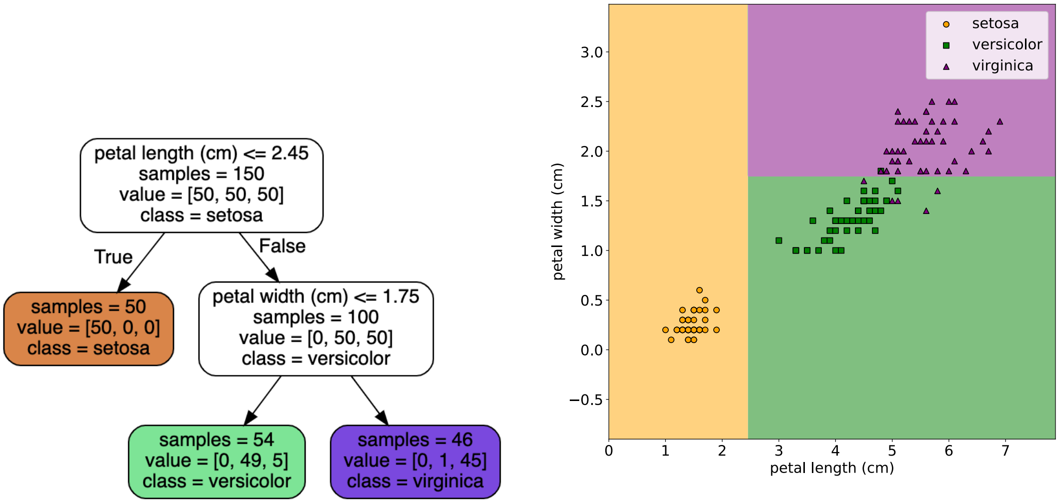
in some other format rather than in a design matrix. However, such data is often converted to a fixed-sized feature representation (a process known as **featurization**), thus implicitly creating a design matrix for further processing. We give an example of this in Section 1.5.4.1, where we discuss the “bag of words” representation for sequence data.

1.2.1.2 Exploratory data analysis

Before tackling a problem with ML, it is usually a good idea to perform **exploratory data analysis**, to see if there are any obvious patterns (which might give hints on what method to choose), or any obvious problems with the data (e.g., label noise or outliers).

For tabular data with a small number of features, it is common to make a **pair plot**, in which panel (i, j) shows a scatter plot of variables i and j, and the diagonal entries (i, i) show the marginal density of variable i; all plots are optionally color coded by class label — see Figure 1.3 for an example.

For higher-dimensional data, it is common to first perform **dimensionality reduction**, and then



(a) *(b)*

Figure 1.4: Example of a decision tree of depth 2 applied to the Iris data, using just the petal length and petal width features. Leaf nodes are color coded according to the predicted class. The number of training samples that pass from the root to a node is shown inside each box; we show how many values of each class fall into this node. This vector of counts can be normalized to get a distribution over class labels for each node. We can then pick the majority class. Adapted from Figures 6.1 and 6.2 of [Gér19]. Generated by iris_ dtree.ipynb.

to visualize the data in 2d or 3d. We discuss methods for dimensionality reduction in Chapter 20.

1.2.1.3 Learning a classifier

From Figure 1.3, we can see that the Setosa class is easy to distinguish from the other two classes. For example, suppose we create the following **decision rule**:

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = \begin{cases} \text{Setosa if petal length} < 2.45 \\ \text{Versicolor or Virginica otherwise} \end{cases} \tag{1.1}$$

This is a very simple example of a classifier, in which we have partitioned the input space into two regions, defined by the one-dimensional (1d) **decision boundary** at $x_{\text{petal length}} = 2.45$. Points lying to the left of this boundary are classified as Setosa; points to the right are either Versicolor or Virginica.

We see that this rule perfectly classifies the Setosa examples, but not the Virginica and Versicolor ones. To improve performance, we can recursively partition the space, by splitting regions in which the classifier makes errors. For example, we can add another decision rule, to be applied to inputs that fail the first test, to check if the petal width is below 1.75cm (in which case we predict Versicolor) or above (in which case we predict Virginica). We can arrange these nested rules into a tree structure,

		Estimate		
		Setosa	Versicolor	Virginica
Truth	Setosa	0	1	1
	Versicolor	1	0	1
	Virginica	10	10	0

Table 1.2: Hypothetical asymmetric loss matrix for Iris classification.

called a **decision tree**, as shown in Figure 1.4a This induces the 2d **decision surface** shown in Figure 1.4b.

We can represent the tree by storing, for each internal node, the feature index that is used, as well as the corresponding threshold value. We denote all these **parameters** by $\boldsymbol{\theta}$. We discuss how to learn these parameters in Section 18.1.

1.2.1.4 Empirical risk minimization

The goal of supervised learning is to automatically come up with classification models such as the one shown in Figure 1.4a, so as to reliably predict the labels for any given input. A common way to measure performance on this task is in terms of the **misclassification rate** on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\left(y_n \neq f(\boldsymbol{x}_n;\boldsymbol{\theta})\right) \tag{1.2}$$

where $\mathbb{I}(e)$ is the binary **indicator function**, which returns 1 iff (if and only if) the condition e is true, and returns 0 otherwise, i.e.,

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases} \tag{1.3}$$

This assumes all errors are equal. However it may be the case that some errors are more costly than others. For example, suppose we are foraging in the wilderness and we find some Iris flowers. Furthermore, suppose that Setosa and Versicolor are tasty, but Virginica is poisonous. In this case, we might use the asymmetric **loss function** $\ell(y, \hat{y})$ shown in Table 1.2.

We can then define **empirical risk** to be the average loss of the predictor on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n;\boldsymbol{\theta})) \tag{1.4}$$

We see that the misclassification rate Equation (1.2) is equal to the empirical risk when we use **zero-one loss** for comparing the true label with the prediction:

$$\ell_{01}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y}) \tag{1.5}$$

See Section 5.1 for more details.

One way to define the problem of **model fitting** or **training** is to find a setting of the parameters that minimizes the empirical risk on the training set:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n;\boldsymbol{\theta})) \tag{1.6}$$

This is called **empirical risk minimization**.

However, our true goal is to minimize the expected loss on *future data* that we have not yet seen. That is, we want to **generalize**, rather than just do well on the training set. We discuss this important point in Section 1.2.3.

1.2.1.5 Uncertainty

> [We must avoid] false confidence bred from an ignorance of the probabilistic nature of the world, from a desire to see black and white where we should rightly see gray. — Immanuel Kant, as paraphrased by Maria Konnikova [Kon20].

In many cases, we will not be able to perfectly predict the exact output given the input, due to lack of knowledge of the input-output mapping (this is called **epistemic uncertainty** or **model uncertainty**), and/or due to intrinsic (irreducible) stochasticity in the mapping (this is called **aleatoric uncertainty** or **data uncertainty**).

Representing uncertainty in our prediction can be important for various applications. For example, let us return to our poisonous flower example, whose loss matrix is shown in Table 1.2. If we predict the flower is Virginica with high probability, then we should not eat the flower. Alternatively, we may be able to perform an **information gathering action**, such as performing a diagnostic test, to reduce our uncertainty. For more information about how to make optimal decisions in the presence of uncertainty, see Section 5.1.

We can capture our uncertainty using the following **conditional probability distribution**:

$$p(y = c|\boldsymbol{x};\boldsymbol{\theta}) = f_c(\boldsymbol{x};\boldsymbol{\theta}) \tag{1.7}$$

where $f : \mathcal{X} \to [0,1]^C$ maps inputs to a probability distribution over the C possible output labels. Since $f_c(\boldsymbol{x};\boldsymbol{\theta})$ returns the probability of class label c, we require $0 \le f_c \le 1$ for each c, and $\sum_{c=1}^{C} f_c = 1$. To avoid this restriction, it is common to instead require the model to return unnormalized log-probabilities. We can then convert these to probabilities using the **softmax function**, which is defined as follows

$$\operatorname{softmax}(\boldsymbol{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^{C} e^{a_{c'}}}, \dots, \frac{e^{a_C}}{\sum_{c'=1}^{C} e^{a_{c'}}}\right] \tag{1.8}$$

This maps $\mathbb{R}^C$ to $[0,1]^C$, and satisfies the constraints that $0 \le \operatorname{softmax}(\boldsymbol{a})_c \le 1$ and $\sum_{c=1}^{C} \operatorname{softmax}(\boldsymbol{a})_c = 1$. The inputs to the softmax, $\boldsymbol{a} = f(\boldsymbol{x};\boldsymbol{\theta})$, are called **logits**. See Section 2.5.2 for details. We thus define the overall model as follows:

$$p(y = c|\boldsymbol{x};\boldsymbol{\theta}) = \operatorname{softmax}_c(f(\boldsymbol{x};\boldsymbol{\theta})) \tag{1.9}$$

A common special case of this arises when f is an **affine function** of the form

$$f(\boldsymbol{x};\boldsymbol{\theta}) = b + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} = b + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D \tag{1.10}$$

where $\boldsymbol{\theta} = (b, \boldsymbol{w})$ are the parameters of the model. This model is called **logistic regression**, and will be discussed in more detail in Chapter 10.

In statistics, the $\boldsymbol{w}$ parameters are usually called **regression coefficients** (and are typically denoted by $\boldsymbol{\beta}$) and b is called the **intercept**. In ML, the parameters $\boldsymbol{w}$ are called the **weights** and b is called the **bias**. This terminology arises from electrical engineering, where we view the function f as a circuit which takes in $\boldsymbol{x}$ and returns $f(\boldsymbol{x})$. Each input is fed to the circuit on "wires", which have weights $\boldsymbol{w}$. The circuit computes the weighted sum of its inputs, and adds a constant bias or offset term b. (This use of the term "bias" should not be confused with the statistical concept of bias discussed in Section 4.7.6.1.)

To reduce notational clutter, it is common to absorb the bias term b into the weights $\boldsymbol{w}$ by defining $\tilde{\boldsymbol{w}} = [b, w_1, \ldots, w_D]$ and defining $\tilde{\boldsymbol{x}} = [1, x_1, \ldots, x_D]$, so that

$$\tilde{\boldsymbol{w}}^\mathsf{T}\tilde{\boldsymbol{x}} = b + \boldsymbol{w}^\mathsf{T}\boldsymbol{x} \tag{1.11}$$

This converts the affine function into a **linear function**. We will usually assume that this has been done, so we can just write the prediction function as follows:

$$f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} \tag{1.12}$$

1.2.1.6 Maximum likelihood estimation

When fitting probabilistic models, it is common to use the negative log probability as our loss function:

$$\ell(y, f(\boldsymbol{x};\boldsymbol{\theta})) = -\log p(y|f(\boldsymbol{x};\boldsymbol{\theta})) \tag{1.13}$$

The reasons for this are explained in Section 5.1.6.1, but the intuition is that a good model (with low loss) is one that assigns a high probability to the true output y for each corresponding input $\boldsymbol{x}$. The average negative log probability of the training set is given by

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N}\log p(y_n|f(\boldsymbol{x}_n;\boldsymbol{\theta})) \tag{1.14}$$

This is called the **negative log likelihood**. If we minimize this, we can compute the **maximum likelihood estimate** or **MLE**:

$$\hat{\boldsymbol{\theta}}_{\text{mle}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\,\text{NLL}(\boldsymbol{\theta}) \tag{1.15}$$

This is a very common way to fit models to data, as we will see.

1.2.2 Regression

Now suppose that we want to predict a real-valued quantity $y \in \mathbb{R}$ instead of a class label $y \in \{1, \ldots, C\}$; this is known as **regression**. For example, in the case of Iris flowers, y might be the degree of toxicity if the flower is eaten, or the average height of the plant.

Regression is very similar to classification. However, since the output is real-valued, we need to use a different loss function. For regression, the most common choice is to use **quadratic loss**, or ℓ_2 **loss**:

$$\ell_2(y, \hat{y}) = (y - \hat{y})^2 \tag{1.16}$$

This penalizes large **residuals** $y - \hat{y}$ more than small ones.[4] The empirical risk when using quadratic loss is equal to the **mean squared error** or **MSE**:

$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - f(\boldsymbol{x}_n; \boldsymbol{\theta}))^2 \tag{1.17}$$

Based on the discussion in Section 1.2.1.5, we should also model the uncertainty in our prediction. In regression problems, it is common to assume the output distribution is a **Gaussian** or **normal**. As we explain in Section 2.6, this distribution is defined by

$$\mathcal{N}(y|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2} \tag{1.18}$$

where μ is the mean, σ^2 is the variance, and $\sqrt{2\pi\sigma^2}$ is the normalization constant needed to ensure the density integrates to 1. In the context of regression, we can make the mean depend on the inputs by defining $\mu = f(\boldsymbol{x}_n; \boldsymbol{\theta})$. We therefore get the following conditional probability distribution:

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}(y|f(\boldsymbol{x}; \boldsymbol{\theta}), \sigma^2) \tag{1.19}$$

If we assume that the variance σ^2 is fixed (for simplicity), the corresponding average (per-sample) negative log likelihood becomes

$$\begin{aligned} \text{NLL}(\boldsymbol{\theta}) &= -\frac{1}{N} \sum_{n=1}^{N} \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_n - f(\boldsymbol{x}_n; \boldsymbol{\theta}))^2 \right) \right] && (1.20) \\ &= \frac{1}{2\sigma^2} \text{MSE}(\boldsymbol{\theta}) + \text{const} && (1.21) \end{aligned}$$

We see that the NLL is proportional to the MSE. Hence computing the maximum likelihood estimate of the parameters will result in minimizing the squared error, which seems like a sensible approach to model fitting.

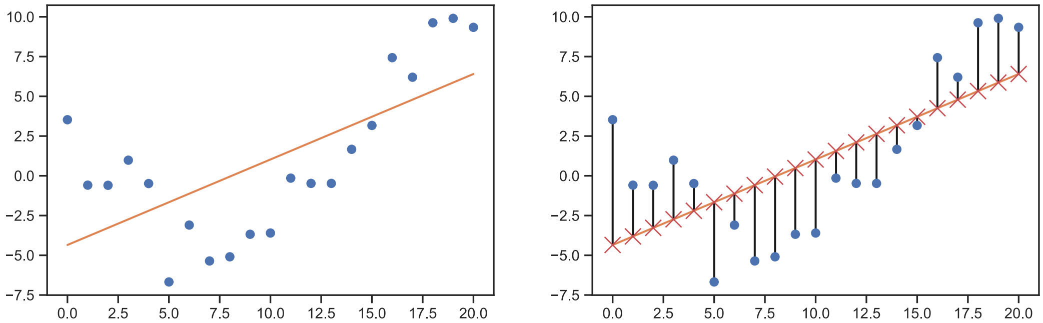
1.2.2.1 Linear regression

As an example of a regression model, consider the 1d data in Figure 1.5a. We can fit this data using a **simple linear regression** model of the form

$$f(x; \boldsymbol{\theta}) = b + wx \tag{1.22}$$

---

[4] If the data has outliers, the quadratic penalty can be too severe. In such cases, it can be better to use ℓ_1 loss instead, which is more **robust**. See Section 11.6 for details.



(a) (b)

Figure 1.5: (a) Linear regression on some 1d data. (b) The vertical lines denote the residuals between the observed output value for each input (blue circle) and its predicted value (red cross). The goal of least squares regression is to pick a line that minimizes the sum of squared residuals. Generated by linreg_residuals_plot.ipynb.

where w is the **slope**, b is the **offset**, and $\boldsymbol{\theta} = (w, b)$ are all the parameters of the model. By adjusting $\boldsymbol{\theta}$, we can minimize the sum of squared errors, shown by the vertical lines in Figure 1.5b. until we find the **least squares solution**

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) \tag{1.23}$$

See Section 11.2.2.1 for details.

If we have multiple input features, we can write

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = b + w_1 x_1 + \cdots + w_D x_D = b + \boldsymbol{w}^\mathsf{T} \boldsymbol{x} \tag{1.24}$$

where $\boldsymbol{\theta} = (\boldsymbol{w}, b)$. This is called **multiple linear regression**.

For example, consider the task of predicting temperature as a function of 2d location in a room. Figure 1.6(a) plots the results of a linear model of the following form:

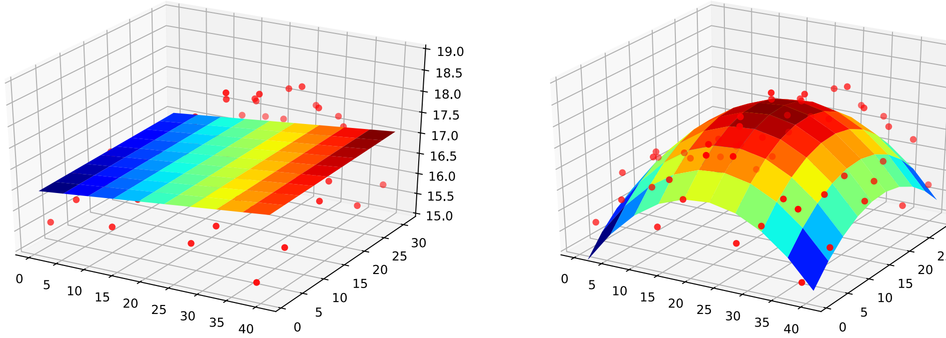
$$f(\boldsymbol{x}; \boldsymbol{\theta}) = b + w_1 x_1 + w_2 x_2 \tag{1.25}$$

We can extend this model to use $D > 2$ input features (such as time of day), but then it becomes harder to visualize.

1.2.2.2 Polynomial regression

The linear model in Figure 1.5a is obviously not a very good fit to the data. We can improve the fit by using a **polynomial regression** model of degree D. This has the form $f(x; \boldsymbol{w}) = \boldsymbol{w}^\mathsf{T} \boldsymbol{\phi}(x)$, where $\boldsymbol{\phi}(x)$ is a feature vector derived from the input, which has the following form:

$$\boldsymbol{\phi}(x) = [1, x, x^2, \ldots, x^D] \tag{1.26}$$



(a)

(b)

Figure 1.6: Linear and polynomial regression applied to 2d data. Vertical axis is temperature, horizontal axes are location within a room. Data was collected by some remote sensing motes at Intel's lab in Berkeley, CA (data courtesy of Romain Thibaux). (a) The fitted plane has the form $\hat{f}(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2$*. (b) Temperature data is fitted with a quadratic of the form* $\hat{f}(\boldsymbol{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2$*. Generated by linreg_2d_surface_demo.ipynb.*

This is a simple example of **feature preprocessing**, also called **feature engineering**.

In Figure 1.7a, we see that using $D = 2$ results in a much better fit. We can keep increasing D, and hence the number of parameters in the model, until $D = N - 1$; in this case, we have one parameter per data point, so we can perfectly **interpolate** the data. The resulting model will have 0 MSE, as shown in Figure 1.7c. However, intuitively the resulting function will not be a good predictor for future inputs, since it is too "wiggly". We discuss this in more detail in Section 1.2.3.

We can also apply polynomial regression to multi-dimensional inputs. For example, Figure 1.6(b) plots the predictions for the temperature model after performing a quadratic expansion of the inputs

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 \tag{1.27}$$

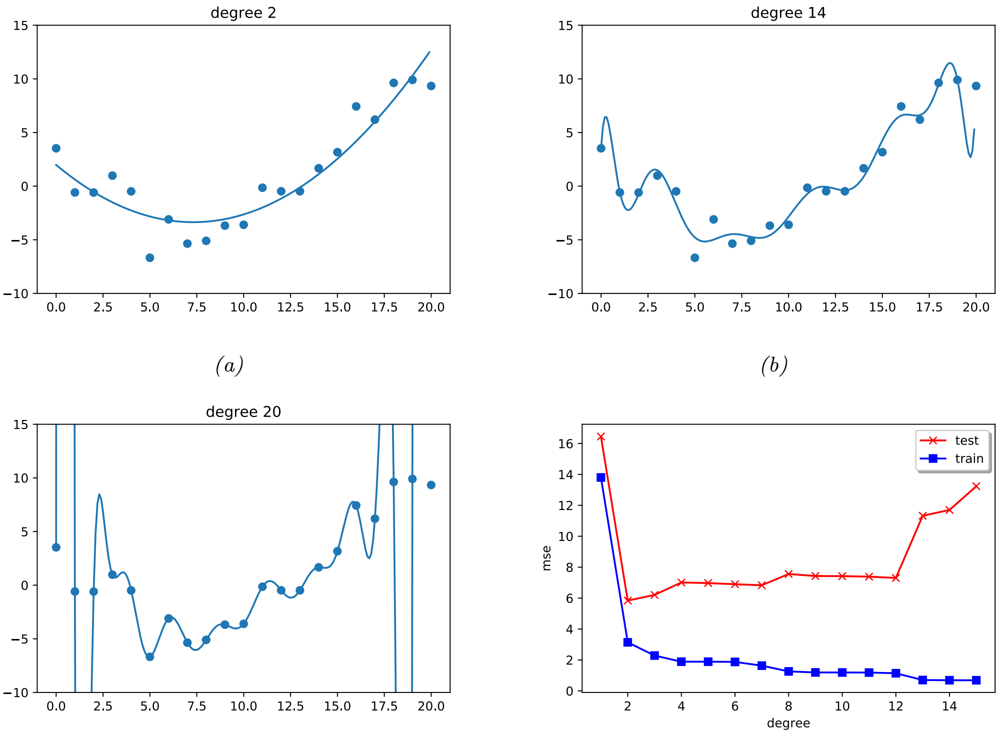
The quadratic shape is a better fit to the data than the linear model in Figure 1.6(a), since it captures the fact that the middle of the room is hotter. We can also add cross terms, such as $x_1 x_2$, to capture interaction effects. See Section 1.5.3.2 for details.

Note that the above models still use a prediction function that is a linear function of the parameters $\boldsymbol{w}$, even though it is a nonlinear function of the original input $\boldsymbol{x}$. The reason this is important is that a linear model induces an MSE loss function $\text{MSE}(\boldsymbol{\theta})$ that has a unique global optimum, as we explain in Section 11.2.2.1.

1.2.2.3 Deep neural networks

In Section 1.2.2.2, we manually specified the transformation of the input features, namely polynomial expansion, $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2, \ldots]$. We can create much more powerful models by learning to do such nonlinear **feature extraction** automatically. If we let $\boldsymbol{\phi}(\boldsymbol{x})$ have its own set of parameters, say $\mathbf{V}$, then the overall model has the form

$$f(\boldsymbol{x}; \boldsymbol{w}, \mathbf{V}) = \boldsymbol{w}^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{x}; \mathbf{V}) \tag{1.28}$$



(a) *(b)*

(c) *(d)*

Figure 1.7: (a-c) Polynomials of degrees 2, 14 and 20 fit to 21 datapoints (the same data as in Figure 1.5). (d) MSE vs degree. Generated by linreg_poly_vs_degree.ipynb.

We can recursively decompose the feature extractor $\boldsymbol{\phi}(\boldsymbol{x}; \mathbf{V})$ into a composition of simpler functions. The resulting model then becomes a stack of L nested functions:

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\boldsymbol{x}))\cdots)) \tag{1.29}$$

where $f_\ell(\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{\theta}_\ell)$ is the function at layer ℓ. The final layer is linear and has the form $f_L(\boldsymbol{x}) = \boldsymbol{w}_L^\mathsf{T}\boldsymbol{x}$, so $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{w}_L^\mathsf{T} f_{1:L-1}(\boldsymbol{x})$, where $f_{1:L-1}(\boldsymbol{x}) = f_{L-1}(\cdots(f_1(\boldsymbol{x}))\cdots)$ is the learned feature extractor. This is the key idea behind **deep neural networks** or **DNN**s, which includes common variants such as **convolutional neural networks** (CNNs) for images, and **recurrent neural networks** (RNNs) for sequences. See Part III for details.

1.2.3 Overfitting and generalization

We can rewrite the empirical risk in Equation (1.4) in the following equivalent way:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}_{\text{train}}} \ell(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{1.30}$$

where $|\mathcal{D}_{\text{train}}|$ is the size of the training set $\mathcal{D}_{\text{train}}$. This formulation is useful because it makes explicit which dataset the loss is being evaluated on.

With a suitably flexible model, we can drive the training loss to zero (assuming no label noise), by simply memorizing the correct output for each input. For example, Figure 1.7(c) perfectly interpolates the training data (modulo the last point on the right). But what we care about is prediction accuracy on new data, which may not be part of the training set. A model that perfectly fits the training data, but which is too complex, is said to suffer from **overfitting**.

To detect if a model is overfitting, let us assume (for now) that we have access to the true (but unknown) distribution $p^*(\boldsymbol{x}, \boldsymbol{y})$ used to generate the training set. Then, instead of computing the empirical risk we compute the theoretical expected loss or **population risk**

$$\mathcal{L}(\boldsymbol{\theta}; p^*) \triangleq \mathbb{E}_{p^*(\boldsymbol{x},\boldsymbol{y})} \left[\ell(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta})) \right] \tag{1.31}$$

The difference $\mathcal{L}(\boldsymbol{\theta}; p^*) - \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}})$ is called the **generalization gap**. If a model has a large generalization gap (i.e., low empirical risk but high population risk), it is a sign that it is overfitting.

In practice we don't know p^*. However, we can partition the data we do have into two subsets, known as the training set and the **test set**. Then we can approximate the population risk using the **test risk**:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) \triangleq \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}_{\text{test}}} \ell(y, f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{1.32}$$

As an example, in Figure 1.7d, we plot the training error and test error for polynomial regression as a function of degree D. We see that the training error goes to 0 as the model becomes more complex. However, the test error has a characteristic **U-shaped curve**: on the left, where $D = 1$, the model is **underfitting**; on the right, where $D \gg 1$, the model is **overfitting**; and when $D = 2$, the model complexity is "just right".

How can we pick a model of the right complexity? If we use the training set to evaluate different models, we will always pick the most complex model, since that will have the most **degrees of freedom**, and hence will have minimum loss. So instead we should pick the model with minimum test loss.

In practice, we need to partition the data into three sets, namely the training set, the test set and a **validation set**; the latter is used for model selection, and we just use the test set to estimate future performance (the population risk), i.e., the test set is not used for model fitting or model selection. See Section 4.5.4 for further details.

1.2.4 No free lunch theorem

> All models are wrong, but some models are useful. — George Box [BD87, p424].[5]

Given the large variety of models in the literature, it is natural to wonder which one is best. Unfortunately, there is no single best model that works optimally for all kinds of problems — this is sometimes called the **no free lunch theorem** [Wol96]. The reason is that a set of assumptions (also called **inductive bias**) that works well in one domain may work poorly in another. The best way to pick a suitable model is based on domain knowledge, and/or trial and error (i.e., using model selection techniques such as cross validation (Section 4.5.4) or Bayesian methods (Section 5.2.2 and Section 5.2.6). For this reason, it is important to have many models and algorithmic techniques in one's toolbox to choose from.

---

[5] George Box is a retired statistics professor at the University of Wisconsin.

1.3 Unsupervised learning

In supervised learning, we assume that each input example $\boldsymbol{x}$ in the training set has an associated set of output targets $\boldsymbol{y}$, and our goal is to learn the input-output mapping. Although this is useful, and can be difficult, supervised learning is essentially just "glorified curve fitting" [Pea18].

An arguably much more interesting task is to try to "make sense of" data, as opposed to just learning a mapping. That is, we just get observed "inputs" $\mathcal{D} = \{\boldsymbol{x}_n : n = 1 : N\}$ without any corresponding "outputs" $\boldsymbol{y}_n$. This is called **unsupervised learning**.

From a probabilistic perspective, we can view the task of unsupervised learning as fitting an unconditional model of the form $p(\boldsymbol{x})$, which can generate new data $\boldsymbol{x}$, whereas supervised learning involves fitting a conditional model, $p(\boldsymbol{y}|\boldsymbol{x})$, which specifies (a distribution over) outputs given inputs.[6]

Unsupervised learning avoids the need to collect large labeled datasets for training, which can often be time consuming and expensive (think of asking doctors to label medical images).

Unsupervised learning also avoids the need to learn how to partition the world into often arbitrary categories. For example, consider the task of labeling when an action, such as "drinking" or "sipping", occurs in a video. Is it when the person picks up the glass, or when the glass first touches the mouth, or when the liquid pours out? What if they pour out some liquid, then pause, then pour again — is that two actions or one? Humans will often disagree on such issues [Idr+17], which means the task is not well defined. It is therefore not reasonable to expect machines to learn such mappings.[7]

Finally, unsupervised learning forces the model to "explain" the high-dimensional inputs, rather than just the low-dimensional outputs. This allows us to learn richer models of "how the world works". As Geoff Hinton, who is a famous professor of ML at the University of Toronto, has said:

> When we're learning to see, nobody's telling us what the right answers are — we just look. Every so often, your mother says "that's a dog", but that's very little information. You'd be lucky if you got a few bits of information — even one bit per second — that way. The brain's visual system has $O(10^{14})$ neural connections. And you only live for $O(10^9)$ seconds. So it's no use learning one bit per second. You need more like $O(10^5)$ bits per second. And there's only one place you can get that much information: from the input itself. — Geoffrey Hinton, 1996 (quoted in [Gor06]).

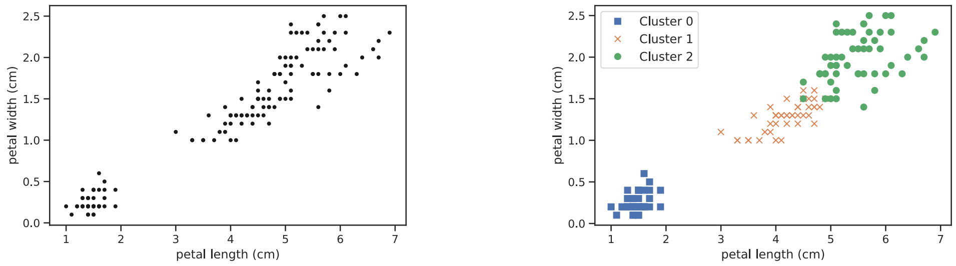
1.3.1 Clustering

A simple example of unsupervised learning is the problem of finding **clusters** in data. The goal is to partition the input into regions that contain "similar" points. As an example, consider a 2d version

---

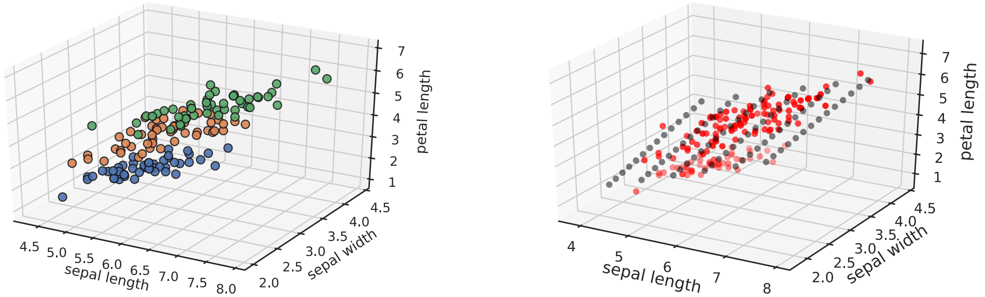
[6] In the statistics community, it is common to use $\boldsymbol{x}$ to denote exogenous variables that are not modeled, but are simply given as inputs. Therefore an unconditional model would be denoted $p(\boldsymbol{y})$ rather than $p(\boldsymbol{x})$.

[7] A more reasonable approach is to try to capture the probability distribution over labels produced by a "crowd" of annotators (see e.g., [Dum+18; Aro+19]). This embraces the fact that there can be multiple "correct" labels for a given input due to the ambiguity of the task itself.



(a) (b)

Figure 1.8: (a) A scatterplot of the petal features from the iris dataset. (b) The result of unsupervised clustering using $K = 3$. Generated by iris_kmeans.ipynb.



(a) (b)

Figure 1.9: (a) Scatterplot of iris data (first 3 features). Points are color coded by class. (b) We fit a 2d linear subspace to the 3d data using PCA. The class labels are ignored. Red dots are the original data, black dots are points generated from the model using $\hat{\boldsymbol{x}} = \mathbf{W}\boldsymbol{z} + \boldsymbol{\mu}$, where $\boldsymbol{z}$ are latent points on the underlying inferred 2d linear manifold. Generated by iris_pca.ipynb.

of the Iris dataset. In Figure 1.8a, we show the points without any class labels. Intuitively there are at least two clusters in the data, one in the bottom left and one in the top right. Furthermore, if we assume that a "good" set of clusters should be fairly compact, then we might want to split the top right into (at least) two subclusters. The resulting partition into three clusters is shown in Figure 1.8b. (Note that there is no correct number of clusters; instead, we need to consider the tradeoff between model complexity and fit to the data. We discuss ways to make this tradeoff in Section 21.3.7.)

1.3.2 Discovering latent "factors of variation"

When dealing with high-dimensional data, it is often useful to reduce the dimensionality by projecting it to a lower dimensional subspace which captures the "essence" of the data. One approach to this problem is to assume that each observed high-dimensional output $\boldsymbol{x}_n \in \mathbb{R}^D$ was generated by a set of hidden or unobserved low-dimensional **latent factors** $\boldsymbol{z}_n \in \mathbb{R}^K$. We can represent the model diagrammatically as follows: $\boldsymbol{z}_n \rightarrow \boldsymbol{x}_n$, where the arrow represents causation. Since we don't know the latent factors $\boldsymbol{z}_n$, we often assume a simple prior probability model for $p(\boldsymbol{z}_n)$ such as a Gaussian, which says that each factor is a random K-dimensional vector. If the data is real-valued, we can use a Gaussian likelihood as well.

The simplest example is when we use a linear model, $p(\boldsymbol{x}_n|\boldsymbol{z}_n;\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_n|\mathbf{W}\boldsymbol{z}_n + \boldsymbol{\mu}, \boldsymbol{\Sigma})$. The resulting model is called **factor analysis** (FA). It is similar to linear regression, except we only observe the outputs $\boldsymbol{x}_n$, and not the inputs $\boldsymbol{z}_n$. In the special case that $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$, this reduces to a model called probabilistic **principal components analysis** (**PCA**), which we will explain in Section 20.1. In Figure 1.9, we give an illustration of how this method can find a 2d linear subspace when applied to some simple 3d data.

Of course, assuming a linear mapping from $\boldsymbol{z}_n$ to $\boldsymbol{x}_n$ is very restrictive. However, we can create nonlinear extensions by defining $p(\boldsymbol{x}_n|\boldsymbol{z}_n;\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_n|f(\boldsymbol{z}_n;\boldsymbol{\theta}), \sigma^2\mathbf{I})$, where $f(\boldsymbol{z};\boldsymbol{\theta})$ is a nonlinear model, such as a deep neural network. It becomes much harder to fit such a model (i.e., to estimate the parameters $\boldsymbol{\theta}$), because the inputs to the neural net have to be inferred, as well as the parameters of the model. However, there are various approximate methods, such as the **variational autoencoder** which can be applied (see Section 20.3.5).

1.3.3 Self-supervised learning

A recently popular approach to unsupervised learning is known as **self-supervised learning**. In this approach, we create proxy supervised tasks from unlabeled data. For example, we might try to learn to predict a color image from a grayscale image, or to mask out words in a sentence and then try to predict them given the surrounding context. The hope is that the resulting predictor $\hat{\boldsymbol{x}}_1 = f(\boldsymbol{x}_2;\boldsymbol{\theta})$, where $\boldsymbol{x}_2$ is the observed input and $\hat{\boldsymbol{x}}_1$ is the predicted output, will learn useful features from the data, that can then be used in standard, downstream supervised tasks. This avoids the hard problem of trying to infer the "true latent factors" $\boldsymbol{z}$ behind the observed data, and instead relies on standard supervised learning methods. We discuss this approach in more detail in Section 19.2.

1.3.4 Evaluating unsupervised learning

Although unsupervised learning is appealing, it is very hard to evaluate the quality of the output of an unsupervised learning method, because there is no ground truth to compare to [TOB16].

A common method for evaluating unsupervised models is to measure the probability assigned by the model to unseen test examples. We can do this by computing the (unconditional) negative log likelihood of the data:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) = -\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) \tag{1.33}$$

This treats the problem of unsupervised learning as one of **density estimation**. The idea is that a good model will not be "surprised" by actual data samples (i.e., will assign them high probability). Furthermore, since probabilities must sum to 1.0, if the model assigns high probability to regions of data space where the data samples come from, it implicitly assigns low probability to the regions where the data does not come from. Thus the model has learned to capture the **typical patterns** in the data. This can be used inside of a **data compression** algorithm.

(a) *(b)*

Figure 1.10: Examples of some control problems. (a) Space Invaders Atari game. From `https://gymnasium.farama.org/environments/atari/space_invaders/`*. (b) Controlling a humanoid robot in the MuJuCo simulator so it walks as fast as possible without falling over. From* `https://gymnasium.farama.org/environments/mujoco/humanoid/`*.*

Unfortunately, density estimation is difficult, especially in high dimensions. Furthermore, a model that assigns high probability to the data may not have learned useful high-level patterns (after all, the model could just memorize all the training examples).

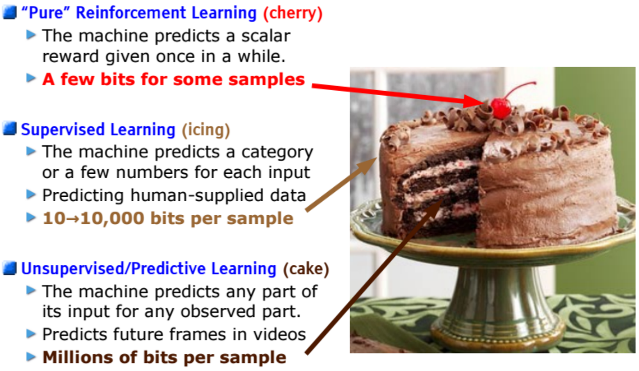
An alternative evaluation metric is to use the learned unsupervised representation as features or input to a downstream supervised learning method. If the unsupervised method has discovered useful patterns, then it should be possible to use these patterns to perform supervised learning using much less labeled data than when working with the original features. For example, in Section 1.2.1.1, we saw how the 4 manually defined features of iris flowers contained most of the information needed to perform classification. We were thus able to train a classifier with nearly perfect performance using just 150 examples. If the input was raw pixels, we would need many more examples to achieve comparable performance (see Section 14.1). That is, we can increase the **sample efficiency** of learning (i.e., reduce the number of labeled examples needed to get good performance) by first learning a good representation.

Increased sample efficiency is a useful evaluation metric, but in many applications, especially in science, the goal of unsupervised learning is to *gain understanding*, not to improve performance on some prediction task. This requires the use of models that are **interpretable**, but which can also generate or "explain" most of the observed patterns in the data. To paraphrase Plato, the goal is to discover how to "carve nature at its joints". Of course, evaluating whether we have successfully discovered the true underlying structure behind some dataset often requires performing experiments and thus interacting with the world. We discuss this topic further in Section 1.4.

1.4 Reinforcement learning

In addition to supervised and unsupervised learning, there is a third kind of ML known as **reinforcement learning** (**RL**). In this class of problems, the system or **agent** has to learn how to interact with its environment. This can be encoded by means of a **policy** $\boldsymbol{a} = \pi(\boldsymbol{x})$, which specifies which action to take in response to each possible input $\boldsymbol{x}$ (derived from the environment state).

For example, consider an agent that learns to play a video game, such as Atari Space Invaders (see



- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning (cake)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

Figure 1.11: The three types of machine learning visualized as layers of a chocolate cake. This figure (originally from `https://bit.ly/2m65Vs1`) was used in a talk by Yann LeCun at NIPS'16, and is used with his kind permission.

Figure 1.10a). In this case, the input $\boldsymbol{x}$ is the image (or sequence of past images), and the output $\boldsymbol{a}$ is the direction to move in (left or right) and whether to fire a missile or not. As a more complex example, consider the problem of a robot learning to walk (see Figure 1.10b). In this case, the input $\boldsymbol{x}$ is the set of joint positions and angles for all the limbs, and the output $\boldsymbol{a}$ is a set of actuation or motor control signals.

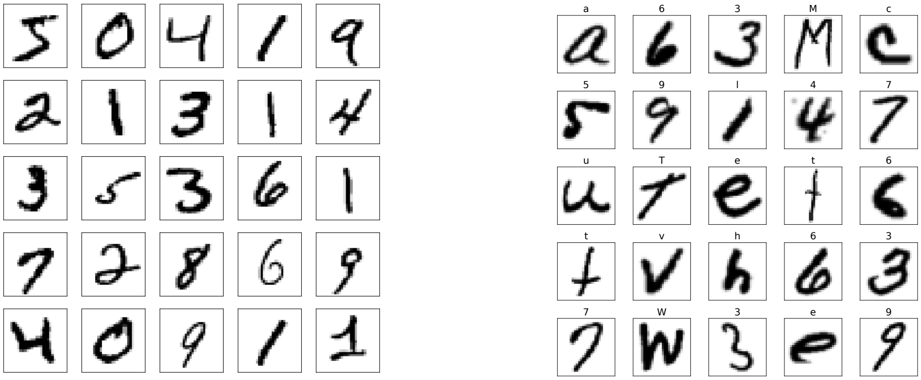
The difference from supervised learning (SL) is that the system is not told which action is the best one to take (i.e., which output to produce for a given input). Instead, the system just receives an occasional **reward** (or punishment) signal in response to the actions that it takes. This is like **learning with a critic**, who gives an occasional thumbs up or thumbs down, as opposed to **learning with a teacher**, who tells you what to do at each step.

RL has grown in popularity recently, due to its broad applicability (since the reward signal that the agent is trying to optimize can be any metric of interest). However, it can be harder to make RL work than it is for supervised or unsupervised learning, for a variety of reasons. A key difficulty is that the reward signal may only be given occasionally (e.g., if the agent eventually reaches a desired state), and even then it may be unclear to the agent which of its many actions were responsible for getting the reward. (Think of playing a game like chess, where there is a single win or lose signal at the end of the game.)

To compensate for the minimal amount of information coming from the reward signal, it is common to use other information sources, such as expert demonstrations, which can be used in a supervised way, or unlabeled data, which can be used by an unsupervised learning system to discover the underlying structure of the environment. This can make it feasible to learn from a limited number of trials (interactions with the environment). As Yann LeCun put it, in an invited talk at the NIPS[8] conference in 2016: "If intelligence was a cake, unsupervised learning would be the chocolate sponge, supervised learning would be the icing, and reinforcement learning would be the cherry." This is illustrated in Figure 1.11.

---

[8] NIPS stands for "Neural Information Processing Systems". It is one of the premier ML conferences. It has recently been renamed to NeurIPS.



(a) *(b)*

Figure 1.12: (a) Visualization of the MNIST dataset. Each image is 28×28*. There are 60k training examples and 10k test examples. We show the first 25 images from the training set. Generated by mnist_viz_tf.ipynb. (b) Visualization of the EMNIST dataset. There are 697,932 training examples, and 116,323 test examples, each of size* 28×28*. There are 62 classes (a-z, A-Z, 0-9). We show the first 25 images from the training set. Generated by emnist_viz_jax.ipynb.*

More information on RL can be found in the sequel to this book, [Mur23].

1.5 Data

Machine learning is concerned with fitting models to data using various algorithms. Although we focus on the modeling and algorithm aspects, it is important to mention that the nature and quality of the training data also plays a vital role in the success of any learned model.

In this section, we briefly describe some common image and text datasets that we will use in this book. We also briefly discuss the topic of data preprocessing.

1.5.1 Some common image datasets

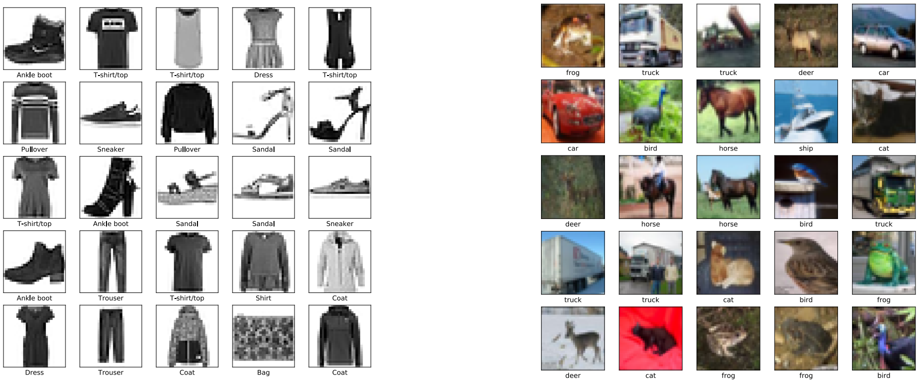
In this section, we briefly discuss some image datasets that we will use in this book.

1.5.1.1 Small image datasets

One of the simplest and most widely used is known as **MNIST** [LeC+98; YB19].[9] This is a dataset of 60k training images and 10k test images, each of size 28×28 (grayscale), illustrating handwritten digits from 10 categories. Each pixel is an integer in the range $\{0, 1, \ldots, 255\}$; these are usually rescaled to $[0, 1]$, to represent pixel intensity. We can optionally convert this to a binary image by thresholding. See Figure 1.12a for an illustration.

---

[9] The term “MNIST” stands for “Modified National Institute of Standards”; The term “modified” is used because the images have been preprocessed to ensure the digits are mostly in the center of the image.


Ankle boot
T-shirt/top
T-shirt/top
Dress
T-shirt/top
Pullover
Sneaker
Pullover
Sandal
Sandal
T-shirt/top
Ankle boot
Sandal
Sandal
Sneaker
Ankle boot
Trouser
T-shirt/top
Shirt
Coat
Dress
Trouser
Coat
Bag
Coat
frog
truck
truck
deer
car
car
bird
horse
ship
cat
deer
horse
horse
bird
truck
truck
truck
cat
bird
frog
deer
cat
frog
frog
bird

(a) (b)

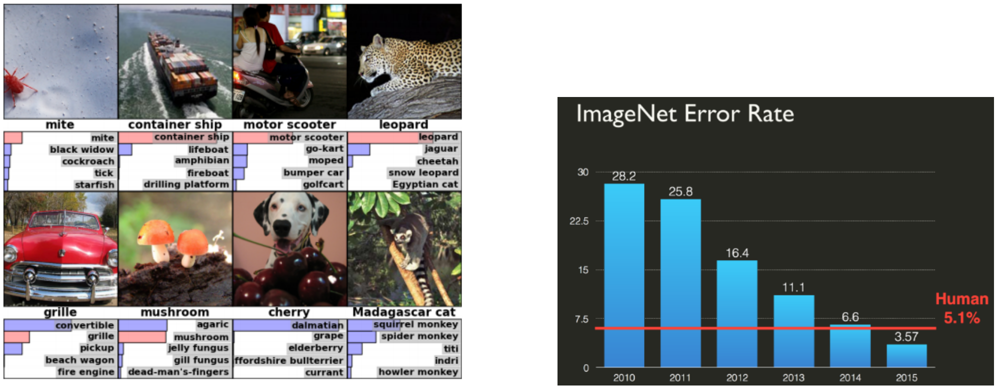
Figure 1.13: (a) Visualization of the Fashion-MNIST dataset [XRV17]. The dataset has the same size as MNIST, but is harder to classify. There are 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle-boot. We show the first 25 images from the training set. Generated by fashion_viz_tf.ipynb. (b) Some images from the CIFAR-10 dataset [KH09]. Each image is $32 \times 32 \times 3$*, where the final dimension of size 3 refers to RGB. There are 50k training examples and 10k test examples. There are 10 classes: plane, car, bird, cat, deer, dog, frog, horse, ship, and truck. We show the first 25 images from the training set. Generated by cifar_viz_tf.ipynb.*

MNIST is so widely used in the ML community that Geoff Hinton, a famous ML researcher, has called it the “drosophila of machine learning”, since if we cannot make a method work well on MNIST, it will likely not work well on harder datasets. However, nowadays MNIST classification is considered “too easy”, since it is possible to distinguish most pairs of digits by looking at just a single pixel. Various extensions have been proposed.

In [Coh+17], they proposed **EMNIST** (extended MNIST), that also includes lower and upper case letters. See Figure 1.12b for a visualization. This dataset is much harder than MNIST, since there are 62 classes, several of which are quite ambiguous (e.g., the digit 1 vs the lower case letter l).

In [XRV17], they proposed **Fashion-MNIST**, which has exactly the same size and shape as MNIST, but where each image is the picture of a piece of clothing instead of a handwritten digit. See Figure 1.13a for a visualization.

For small color images, the most common dataset is **CIFAR** [KH09].[10] This is a dataset of $60k$ images, each of size $32 \times 32 \times 3$, representing everyday objects from 10 or 100 classes; see Figure 1.13b for an illustration.[11]



(a) (b)

Figure 1.14: (a) Sample images from the **ImageNet** *dataset [Rus+15]. This subset consists of 1.3M color training images, each of which is* 256×256 *pixels in size. There are 1000 possible labels, one per image, and the task is to minimize the top-5 error rate, i.e., to ensure the correct label is within the 5 most probable predictions. Below each image we show the true label, and a distribution over the top 5 predicted labels. If the true label is in the top 5, its probability bar is colored red. Predictions are generated by a convolutional neural network (CNN) called "AlexNet" (Section 14.3.2). From Figure 4 of [KSH12]. Used with kind permission of Alex Krizhevsky. (b) Misclassification rate (top 5) on the ImageNet competition over time. Used with kind permission of Andrej Karpathy.*

1.5.1.2 ImageNet

Small datasets are useful for prototyping ideas, but it is also important to test methods on larger datasets, both in terms of image size and number of labeled examples. The most widely used dataset of this type is called **ImageNet** [Rus+15]. This is a dataset of $\sim 14M$ images of size $256 \times 256 \times 3$ illustrating various objects from 20,000 classes; see Figure 1.14a for some examples.

The ImageNet dataset was used as the basis of the ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**), which ran from 2010 to 2018. This used a subset of 1.3M images from 1000 classes. During the course of the competition, significant progress was made by the community, as shown in Figure 1.14b. In particular, 2015 marked the first year in which CNNs could outperform humans (or at least one human, namely Andrej Karpathy) at the task of classifying images from ImageNet. Note that this does not mean that CNNs are better at vision than humans (see e.g., [YL21] for some common failure modes). Instead, it mostly likely reflects the fact that the dataset makes many **fine-grained classification** distinctions — such as between a "tiger" and a "tiger cat" — that humans find difficult to understand; by contrast, sufficiently flexible CNNs can learn arbitrary patterns, including random labels [Zha+17a].

---

10. CIFAR stands for "Canadian Institute For Advanced Research". This is the agency that funded labeling of the dataset, which was derived from the TinyImages dataset at `http://groups.csail.mit.edu/vision/TinyImages/` created by Antonio Torralba. See [KH09] for details.

11. Despite its popularity, the CIFAR dataset has some issues. For example, the base error on CIFAR-100 is 5.85% due to mislabeling [NAM21]. This makes any results with accuracy above 94.15% acc suspicious. Also, 10% of CIFAR-100 training set images are duplicated in the test set [BD20].

1. this film was just brilliant casting location scenery story direction everyone's really suited the part they played robert <UNK> is an amazing actor ...
2. big hair big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horror movies and i've seen hundreds...

Table 1.3: We show snippets of the first two sentences from the IMDB movie review dataset. The first example is labeled positive and the second negative. (`<UNK>` refers to an unknown token.)

Although ImageNet is much harder than MNIST and CIFAR as a classification benchmark, it too is almost "saturated" [Bey+20]. Nevertheless, relative performance of methods on ImageNet is often a surprisingly good predictor of performance on other, unrelated image classification tasks (see e.g., [Rec+19]), so it remains very widely used.

1.5.2 Some common text datasets

Machine learning is often applied to text to solve a variety of tasks. This is known as **natural language processing** or **NLP** (see e.g., [JM20] for details). Below we briefly mention a few text datasets that we will use in this book.

1.5.2.1 Text classification

A simple NLP task is text classification, which can be used for **email spam classification**, **sentiment analysis** (e.g., is a movie or product review positive or negative), etc. A common dataset for evaluating such methods is the **IMDB movie review dataset** from [Maa+11]. (IMDB stands for "Internet Movie Database".) This contains 25k labeled examples for training, and 25k for testing. Each example has a binary label, representing a positive or negative rating. See Table 1.3 for some example sentences.

1.5.2.2 Machine translation

A more difficult NLP task is to learn to map a sentence $\boldsymbol{x}$ in one language to a "semantically equivalent" sentence $\boldsymbol{y}$ in another language; this is called **machine translation**. Training such models requires aligned $(\boldsymbol{x}, \boldsymbol{y})$ pairs. Fortunately, several such datasets exist, e.g., from the Canadian parliament (English-French pairs), and the European Union (Europarl). A subset of the latter, known as the **WMT dataset** (Workshop on Machine Translation), consists of English-German pairs, and is widely used as a benchmark dataset.

1.5.2.3 Other seq2seq tasks

A generalization of machine translation is to learn a mapping from one sequence $\boldsymbol{x}$ to any other sequence $\boldsymbol{y}$. This is called a **seq2seq model**, and can be viewed as a form of high-dimensional classification (see Section 15.2.3 for details). This framing of the problem is very general, and includes many tasks, such as **document summarization**, **question answering**, etc. For example, Table 1.4 shows how to formulate question answering as a seq2seq problem: the input is the text T

T: In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

Q1: What causes precipitation to fall? A1: **gravity**
Q2: What is another main form of precipitation besides drizzle, rain, snow, sleet and hail? A2: **graupel**
Q3: Where do water droplets collide with ice crystals to form precipitation? A3: **within a cloud**

Table 1.4: Question-answer pairs for a sample passage in the SQuAD dataset. Each of the answers is a segment of text from the passage. This can be solved using sentence pair tagging. The input is the paragraph text T and the question Q. The output is a tagging of the relevant words in T that answer the question in Q. From Figure 1 of [Raj+16]. Used with kind permission of Percy Liang.

and question Q, and the output is the answer A, which is a set of words, possibly extracted from the input.

1.5.2.4 Language modeling

The rather grandiose term "**language modeling**" refers to the task of creating unconditional generative models of text sequences, $p(x_1, \ldots, x_T)$. This only requires input sentences $\boldsymbol{x}$, without any corresponding "labels" $\boldsymbol{y}$. We can therefore think of this as a form of unsupervised learning, which we discuss in Section 1.3. If the language model generates output in response to an input, as in seq2seq, we can regard it as a conditional generative model.

1.5.3 Preprocessing discrete input data

Many ML models assume that the data consists of real-valued feature vectors, $\boldsymbol{x} \in \mathbb{R}^D$. However, sometimes the input may have discrete input features, such as categorical variables like race and gender, or words from some vocabulary. In the sections below, we discuss some ways to preprocess such data to convert it to vector form. This is a common operation that is used for many different kinds of models.

1.5.3.1 One-hot encoding

When we have categorical features, we need to convert them to a numerical scale, so that computing weighted combinations of the inputs makes sense. The standard way to preprocess such categorical variables is to use a **one-hot encoding**, also called a **dummy encoding**. If a variable x has K values, we will denote its dummy encoding as follows: $\text{one-hot}(x) = [\mathbb{I}(x = 1), \ldots, \mathbb{I}(x = K)]$. For example, if there are 3 colors (say red, green and blue), the corresponding one-hot vectors will be $\text{one-hot}(\text{red}) = [1, 0, 0]$, $\text{one-hot}(\text{green}) = [0, 1, 0]$, and $\text{one-hot}(\text{blue}) = [0, 0, 1]$.

1.5.3.2 Feature crosses

A linear model using a dummy encoding for each categorical variable can capture the **main effects** of each variable, but cannot capture **interaction effects** between them. For example, suppose we want to predict the fuel efficiency of a vehicle given two categorical input variables: the type (say SUV, Truck, or Family car), and the country of origin (say USA or Japan). If we concatenate the one-hot encodings for the ternary and binary features, we get the following input encoding:

$$\boldsymbol{\phi}(\boldsymbol{x}) = [1, \mathbb{I}(x_1 = S), \mathbb{I}(x_1 = T), \mathbb{I}(x_1 = F), \mathbb{I}(x_2 = U), \mathbb{I}(x_2 = J)] \tag{1.34}$$

where x_1 is the type and x_2 is the country of origin.

This model cannot capture dependencies between the features. For example, we expect trucks to be less fuel efficient, but perhaps trucks from the USA are even less efficient than trucks from Japan. This cannot be captured using the linear model in Equation (1.34) since the contribution from the country of origin is independent of the car type.

We can fix this by computing explicit **feature crosses**. For example, we can define a new composite feature with 3 × 2 possible values, to capture the interaction of type and country of origin. The new model becomes

$$\begin{aligned}
f(\boldsymbol{x}; \boldsymbol{w}) &= \boldsymbol{w}^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{x}) \qquad (1.35)\\
&= w_0 + w_1 \mathbb{I}(x_1 = S) + w_2 \mathbb{I}(x_1 = T) + w_3 \mathbb{I}(x_1 = F)\\
&+ w_4 \mathbb{I}(x_2 = U) + w_5 \mathbb{I}(x_2 = J)\\
&+ w_6 \mathbb{I}(x_1 = S, x_2 = U) + w_7 \mathbb{I}(x_1 = T, x_2 = U) + w_8 \mathbb{I}(x_1 = F, x_2 = U)\\
&+ w_9 \mathbb{I}(x_1 = S, x_2 = J) + w_{10} \mathbb{I}(x_1 = T, x_2 = J) + w_{11} \mathbb{I}(x_1 = F, x_2 = J) \qquad (1.36)
\end{aligned}$$

We can see that the use of feature crosses converts the original dataset into a **wide format**, with many more columns.

1.5.4 Preprocessing text data

In Section 1.5.2, we briefly discussed text classification and other NLP tasks. To feed text data into a classifier, we need to tackle various issues. First, documents have a variable length, and are thus not fixed-length feature vectors, as assumed by many kinds of models. Second, words are categorical variables with many possible values (equal to the size of the vocabulary), so the corresponding one-hot encodings will be very high-dimensional, with no natural notion of similarity. Third, we may encounter words at test time that have not been seen during training (so-called **out-of-vocabulary** or **OOV** words). We discuss some solutions to these problems below. More details can be found in e.g., [BKL10; MRS08; JM20].

1.5.4.1 Bag of words model

A simple approach to dealing with variable-length text documents is to interpret them as a **bag of words**, in which we ignore word order. To convert this to a vector from a fixed input space, we first map each word to a **token** from some vocabulary.

To reduce the number of tokens, we often use various pre-processing techniques such as the following: dropping punctuation, converting all words to lower case; dropping common but uninformative words, such as "and" and "the" (this is called **stop word removal**); replacing words with their base form, such as replacing "running" and "runs" with "run" (this is called **word stemming**); etc. For details, see e.g., [BL12], and for some sample code, see text_preproc_jax.ipynb.

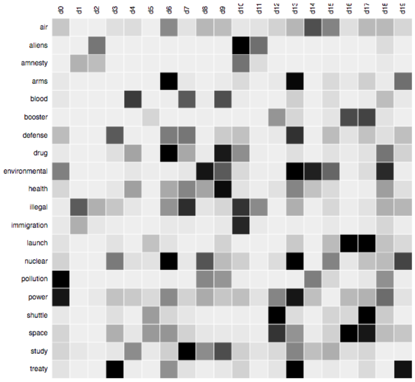


Figure 1.15: Example of a term-document matrix, where raw counts have been replaced by their TF-IDF values (see Section 1.5.4.2). Darker cells are larger values. From `https://bit.ly/2kByLQI`. Used with kind permission of Christoph Carl Kling.

Let x_{nt} be the token at location t in the n'th document. If there are D unique tokens in the vocabulary, then we can represent the n'th document as a D-dimensional vector $\tilde{\boldsymbol{x}}_n$, where $\tilde{x}_{nv}$ is the number of times that word v occurs in document n:

$$\tilde{x}_{nv} = \sum_{t=1}^{T} \mathbb{I}\left(x_{nt} = v\right) \tag{1.37}$$

where T is the length of document n. We can now interpret documents as vectors in $\mathbb{R}^D$. This is called the **vector space model** of text [SWY75; TP10].

We traditionally store input data in an $N \times D$ design matrix denoted by $\mathbf{X}$, where D is the number of features. In the context of vector space models, it is more common to represent the input data as a $D \times N$ **term frequency matrix**, where TF_{ij} is the frequency of term i in document j. See Figure 1.15 for an illustration.

1.5.4.2 TF-IDF

One problem with representing documents as word count vectors is that frequent words may have undue influence, just because the magnitude of their word count is higher, even if they do not carry much semantic content. A common solution to this is to transform the counts by taking logs, which reduces the impact of words that occur many times within a single document.

To reduce the impact of words that occur many times in general (across all documents), we compute a quantity called the **inverse document frequency**, defined as follows: $\mathrm{IDF}_i \triangleq \log \frac{N}{1+\mathrm{DF}_i}$, where DF_i is the number of documents with term i. We can combine these transformations to compute the **TF-IDF** matrix as follows:

$$\mathrm{TFIDF}_{ij} = \log(\mathrm{TF}_{ij} + 1) \times \mathrm{IDF}_i \tag{1.38}$$

(We often normalize each row as well.) This provides a more meaningful representation of documents, and can be used as input to many ML algorithms. See tfidf_demo.ipynb for an example.

1.5.4.3 Word embeddings

Although the TF-IDF transformation improves on raw count vectors by placing more weight on “informative” words and less on “uninformative” words, it does not solve the fundamental problem with the one-hot encoding (from which count vectors are derived), which is that that semantically similar words, such as “man” and “woman”, may be not be any closer (in vector space) than semantically dissimilar words, such as “man” and “banana”. Thus the assumption that points that are close in input space should have similar outputs, which is implicitly made by most prediction models, is invalid.

The standard way to solve this problem is to use **word embeddings**, in which we map each sparse one-hot vector, $\boldsymbol{x}_{nt} \in \{0,1\}^V$, to a lower-dimensional dense vector, $\boldsymbol{e}_{nt} \in \mathbb{R}^K$ using $\boldsymbol{e}_{nt} = \mathbf{E}\boldsymbol{x}_{nt}$, where $\mathbf{E} \in \mathbb{R}^{K \times V}$ is learned such that semantically similar words are placed close by. There are many ways to learn such embeddings, as we discuss in Section 20.5.

Once we have an embedding matrix, we can represent a variable-length text document as a **bag of word embeddings**. We can then convert this to a fixed length vector by summing (or averaging) the embeddings:

$$\overline{\boldsymbol{e}}_n = \sum_{t=1}^{T} \boldsymbol{e}_{nt} = \mathbf{E}\tilde{\boldsymbol{x}}_n \tag{1.39}$$

where $\tilde{\boldsymbol{x}}_n$ is the bag of words representation from Equation (1.37). We can then use this inside of a logistic regression classifier, which we briefly introduced in Section 1.2.1.5. The overall model has the form

$$p(y = c|\boldsymbol{x}_n, \boldsymbol{\theta}) = \text{softmax}_c(\mathbf{W}\mathbf{E}\tilde{\boldsymbol{x}}_n) \tag{1.40}$$

We often use a **pre-trained word embedding** matrix $\mathbf{E}$, in which case the model is linear in $\mathbf{W}$, which simplifies parameter estimation (see Chapter 10). See also Section 15.7 for a discussion of contextual word embeddings.

1.5.4.4 Dealing with novel words

At test time, the model may encounter a completely novel word that it has not seen before. This is known as the **out of vocabulary** or **OOV** problem. Such novel words are bound to occur, because the set of words is an **open class**. For example, the set of proper nouns (names of people and places) is unbounded.

A standard heuristic to solve this problem is to replace all novel words with the special symbol **UNK**, which stands for “unknown”. However, this loses information. For example, if we encounter the word “athazagoraphobia”, we may guess it means “fear of something”, since phobia is a common suffix in English (derived from Greek) to mean “fear of”. (It turns out that athazagoraphobia means “fear of being forgotten about or ignored”.)

We could work at the character level, but this would require the model to learn how to group common letter combinations together into words. It is better to leverage the fact that words have substructure, and then to take as input **subword units** or **wordpieces** [SHB16; Wu+16]; these are often created using a method called **byte-pair encoding** [Gag94], which is a form of data compression that creates new symbols to represent common substrings.

1.5.5 Handling missing data

Sometimes we may have **missing data**, in which parts of the input $\boldsymbol{x}$ or output y may be unknown. If the output is unknown during training, the example is unlabeled; we consider such semi-supervised learning scenarios in Section 19.3. We therefore focus on the case where some of the input features may be missing, either at training or testing time, or both.

To model this, let $\mathbf{M}$ be an $N \times D$ matrix of binary variables, where $M_{nd} = 1$ if feature d in example n is missing, and $M_{nd} = 0$ otherwise. Let $\mathbf{X}_v$ be the visible parts of the input feature matrix, corresponding to $M_{nd} = 0$, and $\mathbf{X}_h$ be the missing parts, corresponding to $M_{nd} = 1$. Let $\mathbf{Y}$ be the output label matrix, which we assume is fully observed. If we assume $p(\mathbf{M}|\mathbf{X}_v, \mathbf{X}_h, \mathbf{Y}) = p(\mathbf{M})$, we say the data is **missing completely at random** or **MCAR**, since the missingness does not depend on the hidden or observed features. If we assume $p(\mathbf{M}|\mathbf{X}_v, \mathbf{X}_h, \mathbf{Y}) = p(\mathbf{M}|\mathbf{X}_v, \mathbf{Y})$, we say the data is **missing at random** or **MAR**, since the missingness does not depend on the hidden features, but may depend on the visible features. If neither of these assumptions hold, we say the data is **not missing at random** or **NMAR**.

In the MCAR and MAR cases, we can ignore the missingness mechanism, since it tells us nothing about the hidden features. However, in the NMAR case, we need to model the **missing data mechanism**, since the lack of information may be informative. For example, the fact that someone did not fill out an answer to a sensitive question on a survey (e.g., "Do you have COVID?") could be informative about the underlying value. See e.g., [LR87; Mar08] for more information on missing data models.

In this book, we will always make the MAR assumption. However, even with this assumption, we cannot directly use a discriminative model, such as a DNN, when we have missing input features, since the input $\boldsymbol{x}$ will have some unknown values.

A common heuristic is called **mean value imputation**, in which missing values are replaced by their empirical mean. More generally, we can fit a generative model to the input, and use that to **fill in** the missing values. We briefly discuss some suitable generative models for this task in Chapter 20, and in more detail in the sequel to this book, [Mur23].

1.6 Discussion

In this section, we situate ML and this book into a larger context.

1.6.1 The relationship between ML and other fields

There are several subcommunities that work on ML-related topics, each of which have different names. The field of **predictive analytics** is similar to supervised learning (in particular, classification and regression), but focuses more on business applications. **Data mining** covers both supervised and unsupervised machine learning, but focuses more on structured data, usually stored in large commercial databases. **Data science** uses techniques from machine learning and statistics, but also emphasizes other topics, such as data integration, data visualization, and working with domain experts, often in an iterative feedback loop (see e.g., [BS17]). The difference between these areas is often just one of terminology.[12]

---

[12] See `https://developers.google.com/machine-learning/glossary/` for a useful "ML glossary".

ML is also very closely related to the field of **statistics**. Indeed, Jerry Friedman, a famous statistics professor at Stanford, said[13]

> [If the statistics field had] incorporated computing methodology from its inception as a fundamental tool, as opposed to simply a convenient way to apply our existing tools, many of the other data related fields [such as ML] would not have needed to exist — they would have been part of statistics. — Jerry Friedman [Fri97b]

Machine learning is also related to **artificial intelligence** (**AI**). Historically, the field of AI assumed that we could program "intelligence" by hand (see e.g., [RN10; PM17]), but this approach has largely failed to live up to expectations, mostly because it proved to be too hard to explicitly encode all the knowledge such systems need. Consequently, there is renewed interest in using ML to help an AI system acquire its own knowledge. (Indeed the connections are so close that sometimes the terms "ML" and "AI" are used interchangeably, although this is arguably misleading [Pre21].)

1.6.2 Structure of the book

We have seen that ML is closely related to many other subjects in mathematics, statistics, computer science, etc. It can be hard to know where to start.

In this book, we take one particular path through this interconnected landscape, using probability theory as our unifying lens. We cover statistical foundations in Part I, supervised learning in Part II–Part IV, and unsupervised learning in Part V. For more information on these (and other) topics, please see the sequel to this book, [Mur23],

In addition to the book, you may find the online Python notebooks that accompany this book helpful. See `probml.github.io/book1` for details.

1.6.3 Caveats

In this book, we will see how machine learning can be used to create systems that can (attempt to) predict outputs given inputs. These predictions can then be used to choose actions so as to minimize expected loss. When designing such systems, it can be hard to design a loss function that correctly specifies all of our preferences; this can result in "**reward hacking**" in which the machine optimizes the reward function we give it, but then we realize that the function did not capture various constraints or preferences that we forgot to specify [Wei76; Amo+16; D'A+20]. (This is particularly important when tradeoffs need to be made between multiple objectives.)

Reward hacking is an example of a larger problem known as the "**alignment problem**" [Chr20], which refers to the potential discrepancy between what we ask our algorithms to optimize and what we actually want them to do for us; this has raised various concerns in the context of **AI ethics** and **AI safety** (see e.g., [KR19; Lia20; Spe+22]). Russell [Rus19] proposes to solve this problem by not explicitly specifying a reward function, but instead forcing the machine to infer the reward by observing human behavior, an approach known as **inverse reinforcement learning**. However, emulating current or past human behavior too closely may be undesirable, and can be biased by the data that is available for training (see e.g., [Pau+20]).

The above view of AI, in which an "intelligent" system makes decisions on its own, without a human in the loop, is believed by many to be the path towards "**artificial general intelligence**" or **AGI**. An alternative approach is to view AI as "**augmented intelligence**" (sometimes called **intelligence augmentation** or **IA**). In this paradigm, AI is a process for creating "smart tools", like adaptive cruise control or auto-complete in search engines; such tools maintain a human in the decision-making loop. In this framing, systems which have AI/ML components in them are not that different from other complex, semi-autonomous human artefacts, such as aeroplanes with autopilot, online trading platforms or medical diagnostic systems (c.f. [Jor19; Ace]). Of course, as the AI tools become more powerful, they can end up doing more and more on their own, making this approach similar to AGI. However, in augmented intelligence, the goal is not to emulate or exceed human behavior at certain tasks, but instead to help humans get stuff done more easily; this is how we treat most other technologies [Kap16].

---

[13] Quoted in `https://brenocon.com/blog/2008/12/statistics-vs-machine-learning-fight/`

Part I

Foundations

2 Probability: Univariate Models

2.1 Introduction

In this chapter, we give a brief introduction to the basics of probability theory. There are many good books that go into more detail, e.g., [GS97; BT08; Cha21].

2.1.1 What is probability?

> Probability theory is nothing but common sense reduced to calculation. — Pierre Laplace, 1812

We are all comfortable saying that the probability that a (fair) coin will land heads is 50%. But what does this mean? There are actually two different interpretations of probability. One is called the **frequentist** interpretation. In this view, probabilities represent long run frequencies of **events** that can happen multiple times. For example, the above statement means that, if we flip the coin many times, we expect it to land heads about half the time.[1]

The other interpretation is called the **Bayesian** interpretation of probability. In this view, probability is used to quantify our **uncertainty** or ignorance about something; hence it is fundamentally related to information rather than repeated trials [Jay03; Lin06]. In the Bayesian view, the above statement means we believe the coin is equally likely to land heads or tails on the next toss.

One big advantage of the Bayesian interpretation is that it can be used to model our uncertainty about one-off events that do not have long term frequencies. For example, we might want to compute the probability that the polar ice cap will melt by 2030 CE. This event will happen zero or one times, but cannot happen repeatedly. Nevertheless, we ought to be able to quantify our uncertainty about this event; based on how probable we think this event is, we can decide how to take the optimal action, as discussed in Chapter 5. We shall therefore adopt the Bayesian interpretation in this book. Fortunately, the basic rules of probability theory are the same, no matter which interpretation is adopted.

2.1.2 Types of uncertainty

The uncertainty in our predictions can arise for two fundamentally different reasons. The first is due to our ignorance of the underlying hidden causes or mechanism generating our data. This is called **epistemic uncertainty**, since epistemology is the philosophical term used to describe the study of knowledge. However, a simpler term for this is **model uncertainty**. The second kind of uncertainty arises from intrinsic variability, which cannot be reduced even if we collect more data. This is sometimes called **aleatoric uncertainty** [Hac75; KD09], derived from the Latin word for "dice", although a simpler term would be **data uncertainty**. As a concrete example, consider tossing a fair coin. We might know for sure that the probability of heads is $p = 0.5$, so there is no epistemic uncertainty, but we still cannot perfectly predict the outcome.

---

[1] Actually, the Stanford statistician (and former professional magician) Persi Diaconis has shown that a coin is about 51% likely to land facing the same way up as it started, due to the physics of the problem [DHM07].

This distinction can be important for applications such as active learning. A typical strategy is to query examples for which $\mathbb{H}(p(y|\boldsymbol{x}, \mathcal{D}))$ is large (where $\mathbb{H}(p)$ is the entropy, discussed in Section 6.1). However, this could be due to uncertainty about the parameters, i.e., large $\mathbb{H}(p(\boldsymbol{\theta}|\mathcal{D}))$, or just due to inherent variability of the outcome, corresponding to large entropy of $p(y|\boldsymbol{x}, \boldsymbol{\theta})$. In the latter case, there would not be much use collecting more samples, since our uncertainty would not be reduced. See [Osb16] for further discussion of this point.

2.1.3 Probability as an extension of logic

In this section, we review the basic rules of probability, following the presentation of [Jay03], in which we view probability as an extension of **Boolean logic**.

2.1.3.1 Probability of an event

We define an **event**, denoted by the binary variable A, as some state of the world that either holds or does not hold. For example, A might be event "it will rain tomorrow", or "it rained yesterday", or "the label is $y = 1$", or "the parameter θ is between 1.5 and 2.0", etc. The expression $\Pr(A)$ denotes the probability with which you believe event A is true (or the long run fraction of times that A will occur). We require that $0 \leq \Pr(A) \leq 1$, where $\Pr(A) = 0$ means the event definitely will not happen, and $\Pr(A) = 1$ means the event definitely will happen. We write $\Pr(\overline{A})$ to denote the probability of event A not happening; this is defined to be $\Pr(\overline{A}) = 1 - \Pr(A)$.

2.1.3.2 Probability of a conjunction of two events

We denote the **joint probability** of events A and B both happening as follows:

$$\Pr(A \wedge B) = \Pr(A, B) \tag{2.1}$$

If A and B are independent events, we have

$$\Pr(A, B) = \Pr(A) \Pr(B) \tag{2.2}$$

For example, suppose X and Y are chosen uniformly at random from the set $\mathcal{X} = \{1, 2, 3, 4\}$. Let A be the event that $X \in \{1, 2\}$, and B be the event that $Y \in \{3\}$. Then we have $\Pr(A, B) = \Pr(A) \Pr(B) = \frac{1}{2} \cdot \frac{1}{4}$.

2.1.3.3 Probability of a union of two events

The probability of event A or B happening is given by

$$\Pr(A \vee B) = \Pr(A) + \Pr(B) - \Pr(A \wedge B) \tag{2.3}$$

If the events are mutually exclusive (so they cannot happen at the same time), we get

$$\Pr(A \vee B) = \Pr(A) + \Pr(B) \tag{2.4}$$

For example, suppose X is chosen uniformly at random from the set $\mathcal{X} = \{1, 2, 3, 4\}$. Let A be the event that $X \in \{1, 2\}$ and B be the event that $X \in \{3\}$. Then we have $\Pr(A \vee B) = \frac{2}{4} + \frac{1}{4}$.

2.1.3.4 Conditional probability of one event given another

We define the **conditional probability** of event B happening given that A has occurred as follows:

$$\Pr(B|A) \triangleq \frac{\Pr(A, B)}{\Pr(A)} \tag{2.5}$$

This is not defined if $\Pr(A) = 0$, since we cannot condition on an impossible event.

2.1.3.5 Independence of events

We say that event A is **independent** of event B if

$$\Pr(A, B) = \Pr(A) \Pr(B) \tag{2.6}$$

2.1.3.6 Conditional independence of events

We say that events A and B are **conditionally independent** given event C if

$$\Pr(A, B|C) = \Pr(A|C) \Pr(B|C) \tag{2.7}$$

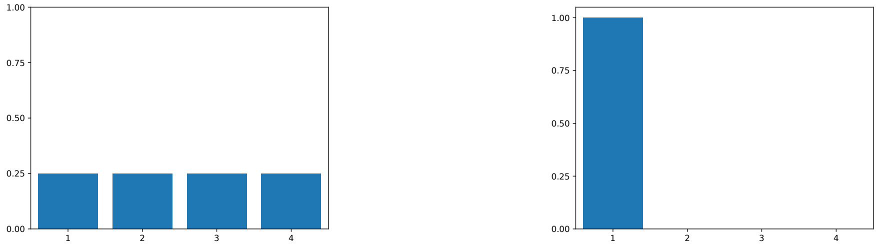
This is written as $A \perp B|C$. Events are often dependent on each other, but may be rendered independent if we condition on the relevant intermediate variables, as we discuss in more detail later in this chapter.

2.2 Random variables

Suppose X represents some unknown quantity of interest, such as which way a dice will land when we roll it, or the temperature outside your house at the current time. If the value of X is unknown and/or could change, we call it a **random variable** or **rv**. The set of possible values, denoted $\mathcal{X}$, is known as the **sample space** or **state space**. An **event** is a set of outcomes from a given sample space. For example, if X represents the face of a dice that is rolled, so $\mathcal{X} = \{1, 2, \ldots, 6\}$, the event of "seeing a 1" is denoted $X = 1$, the event of "seeing an odd number" is denoted $X \in \{1, 3, 5\}$, the event of "seeing a number between 1 and 3" is denoted $1 \leq X \leq 3$, etc.

2.2.1 Discrete random variables

If the sample space $\mathcal{X}$ is finite or countably infinite, then X is called a **discrete random variable**. In this case, we denote the probability of the event that X has value x by $\Pr(X = x)$. We define the



(a) (b)

Figure 2.1: Some discrete distributions on the state space $\mathcal{X} = \{1, 2, 3, 4\}$. (a) A uniform distribution with $p(x = k) = 1/4$. (b) A degenerate distribution (delta function) that puts all its mass on $x = 1$. Generated by discrete_prob_dist_plot.ipynb.

probability mass function or **pmf** as a function which computes the probability of events which correspond to setting the rv to each possible value:

$$p(x) \triangleq \Pr(X = x) \tag{2.8}$$

The pmf satisfies the properties $0 \leq p(x) \leq 1$ and $\sum_{x \in \mathcal{X}} p(x) = 1$.

If X has a finite number of values, say K, the pmf can be represented as a list of K numbers, which we can plot as a histogram. For example, Figure 2.1 shows two pmf's defined on $\mathcal{X} = \{1, 2, 3, 4\}$. On the left we have a uniform distribution, $p(x) = 1/4$, and on the right, we have a degenerate distribution, $p(x) = \mathbb{I}(x = 1)$, where $\mathbb{I}()$ is the binary indicator function. Thus the distribution in Figure 2.1(b) represents the fact that X is always equal to the value 1. (Thus we see that random variables can also be constant.)

2.2.2 Continuous random variables

If $X \in \mathbb{R}$ is a real-valued quantity, it is called a **continuous random variable**. In this case, we can no longer create a finite (or countable) set of distinct possible values it can take on. However, there are a countable number of *intervals* which we can partition the real line into. If we associate events with X being in each one of these intervals, we can use the methods discussed above for discrete random variables. Informally speaking, we can represent the probability of X taking on a specific real value by allowing the size of the intervals to shrink to zero, as we show below.

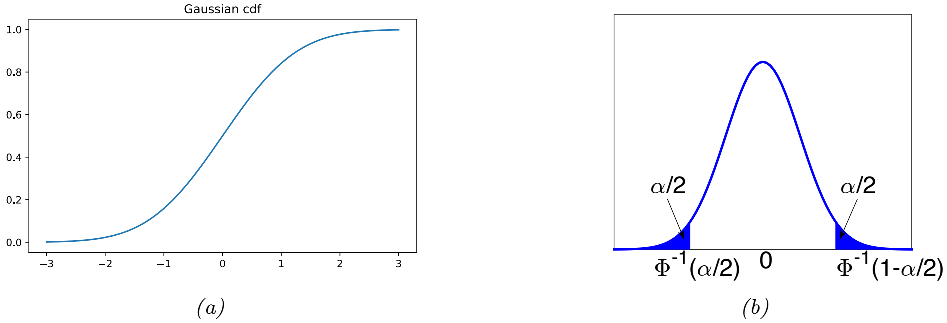
2.2.2.1 Cumulative distribution function (cdf)

Define the events $A = (X \leq a)$, $B = (X \leq b)$ and $C = (a < X \leq b)$, where $a < b$. We have that $B = A \vee C$, and since A and C are mutually exclusive, the sum rules gives

$$\Pr(B) = \Pr(A) + \Pr(C) \tag{2.9}$$

and hence the probability of being in interval C is given by

$$\Pr(C) = \Pr(B) - \Pr(A) \tag{2.10}$$



(a) (b)

Figure 2.2: (a) Plot of the cdf for the standard normal, $\mathcal{N}(0, 1)$. Generated by gauss_plot.ipynb. (b) Corresponding pdf. The shaded regions each contain $\alpha/2$ of the probability mass. Therefore the nonshaded region contains $1 - \alpha$ of the probability mass. The leftmost cutoff point is $\Phi^{-1}(\alpha/2)$, where Φ is the cdf of the Gaussian. By symmetry, the rightmost cutoff point is $\Phi^{-1}(1 - \alpha/2) = -\Phi^{-1}(\alpha/2)$. Generated by quantile_plot.ipynb.

In general, we define the **cumulative distribution function** or **cdf** of the rv X as follows:

$$P(x) \triangleq \Pr(X \le x) \tag{2.11}$$

(Note that we use a capital P to represent the cdf.) Using this, we can compute the probability of being in any interval as follows:

$$\Pr(a < X \le b) = P(b) - P(a) \tag{2.12}$$

Cdf's are monotonically non-decreasing functions. See Figure 2.2a for an example, where we illustrate the cdf of a standard normal distribution, $\mathcal{N}(x|0, 1)$; see Section 2.6 for details.

2.2.2.2 Probability density function (pdf)

We define the **probability density function** or **pdf** as the derivative of the cdf:

$$p(x) \triangleq \frac{d}{dx} P(x) \tag{2.13}$$

(Note that this derivative does not always exist, in which case the pdf is not defined.) See Figure 2.2b for an example, where we illustrate the pdf of a univariate Gaussian (see Section 2.6 for details).

Given a pdf, we can compute the probability of a continuous variable being in a finite interval as follows:

$$\Pr(a < X \le b) = \int_a^b p(x) dx = P(b) - P(a) \tag{2.14}$$

As the size of the interval gets smaller, we can write

$$\Pr(x < X \le x + dx) \approx p(x) dx \tag{2.15}$$

Intuitively, this says the probability of X being in a small interval around x is the density at x times the width of the interval.

2.2.2.3 Quantiles

If the cdf P is strictly monotonically increasing, it has an inverse, called the **inverse cdf**, or **percent point function** (**ppf**), or **quantile function**.

If P is the cdf of X, then $P^{-1}(q)$ is the value x_q such that $\Pr(X \leq x_q) = q$; this is called the q'th **quantile** of P. The value $P^{-1}(0.5)$ is the **median** of the distribution, with half of the probability mass on the left, and half on the right. The values $P^{-1}(0.25)$ and $P^{-1}(0.75)$ are the lower and upper **quartiles**.

For example, let Φ be the cdf of the Gaussian distribution $\mathcal{N}(0,1)$, and Φ^{-1} be the inverse cdf. Then points to the left of $\Phi^{-1}(\alpha/2)$ contain $\alpha/2$ of the probability mass, as illustrated in Figure 2.2b. By symmetry, points to the right of $\Phi^{-1}(1-\alpha/2)$ also contain $\alpha/2$ of the mass. Hence the central interval $(\Phi^{-1}(\alpha/2), \Phi^{-1}(1-\alpha/2))$ contains $1-\alpha$ of the mass. If we set $\alpha = 0.05$, the central 95% interval is covered by the range

$$(\Phi^{-1}(0.025), \Phi^{-1}(0.975)) = (-1.96, 1.96) \tag{2.16}$$

If the distribution is $\mathcal{N}(\mu, \sigma^2)$, then the 95% interval becomes $(\mu - 1.96\sigma, \mu + 1.96\sigma)$. This is often approximated by writing $\mu \pm 2\sigma$.

2.2.3 Sets of related random variables

In this section, we discuss distributions over sets of related random variables.

Suppose, to start, that we have two random variables, X and Y. We can define the **joint distribution** of two random variables using $p(x, y) = p(X = x, Y = y)$ for all possible values of X and Y. If both variables have finite cardinality, we can represent the joint distribution as a 2d table, all of whose entries sum to one. For example, consider the following example with two binary variables:

$p(X,Y)$	$Y=0$	$Y=1$
$X=0$	0.2	0.3
$X=1$	0.3	0.2

If two variables are independent, we can represent the joint as the product of the two marginals. If both variables have finite cardinality, we can factorize the 2d joint table into a product of two 1d vectors, as shown in Figure 2.3.

Given a joint distribution, we define the **marginal distribution** of an rv as follows:

$$p(X = x) = \sum_y p(X = x, Y = y) \tag{2.17}$$

where we are summing over all possible states of Y. This is sometimes called the **sum rule** or the **rule of total probability**. We define $p(Y = y)$ similarly. For example, from the above 2d table, we see $p(X = 0) = 0.2 + 0.3 = 0.5$ and $p(Y = 0) = 0.2 + 0.3 = 0.5$. (The term "marginal" comes from the accounting practice of writing the sums of rows and columns on the side, or margin, of a table.)

We define the **conditional distribution** of an rv using

$$p(Y = y|X = x) = \frac{p(X = x, Y = y)}{p(X = x)} \tag{2.18}$$

We can rearrange this equation to get

$$p(x, y) = p(x)p(y|x) \tag{2.19}$$

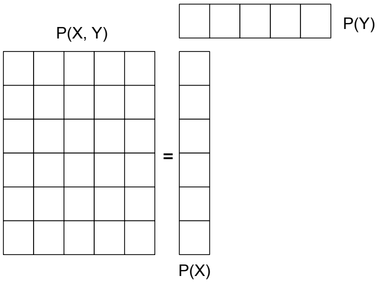


Figure 2.3: Computing $p(x, y) = p(x)p(y)$, where $X \perp Y$. Here X and Y are discrete random variables; X has 6 possible states (values) and Y has 5 possible states. A general joint distribution on two such variables would require $(6 \times 5) - 1 = 29$ parameters to define it (we subtract 1 because of the sum-to-one constraint). By assuming (unconditional) independence, we only need $(6 - 1) + (5 - 1) = 9$ parameters to define $p(x, y)$.

This is called the **product rule**.

By extending the product rule to D variables, we get the **chain rule of probability**:

$$p(\boldsymbol{x}_{1:D}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_1, x_2, x_3) \dots p(x_D|\boldsymbol{x}_{1:D-1}) \tag{2.20}$$

This provides a way to create a high dimensional joint distribution from a set of conditional distributions. We discuss this in more detail in Section 3.6.

2.2.4 Independence and conditional independence

We say X and Y are **unconditionally independent** or **marginally independent**, denoted $X \perp Y$, if we can represent the joint as the product of the two marginals (see Figure 2.3), i.e.,

$$X \perp Y \iff p(X, Y) = p(X)p(Y) \tag{2.21}$$

In general, we say a set of variables $X_1, \dots, X_n$ is (mutually) **independent** if the joint can be written as a product of marginals for all subsets $\{X_1, \dots, X_m\} \subseteq \{X_1, \dots, X_n\}$: i.e.,

$$p(X_1, \dots, X_m) = \prod_{i=1}^{m} p(X_i) \tag{2.22}$$

For example, we say X_1, X_2, X_3 are mutually independent if the following conditions hold: $p(X_1, X_2, X_3) = p(X_1)p(X_2)p(X_3)$, $p(X_1, X_2) = p(X_1)p(X_2)$, $p(X_2, X_3) = p(X_2)p(X_3)$, and $p(X_1, X_3) = p(X_1)p(X_3)$.[2]

Unfortunately, unconditional independence is rare, because most variables can influence most other variables. However, usually this influence is mediated via other variables rather than being direct. We therefore say X and Y are **conditionally independent** (CI) given Z iff the conditional joint can be written as a product of conditional marginals:

$$X \perp Y \mid Z \iff p(X, Y|Z) = p(X|Z)p(Y|Z) \tag{2.23}$$

---

[2] For further discussion, see `https://github.com/probml/pml-book/issues/353#issuecomment-1120327442`.

We can write this assumption as a graph $X - Z - Y$, which captures the intuition that all the dependencies between X and Y are mediated via Z. By using larger graphs, we can define complex joint distributions; these are known as **graphical models**, and are discussed in Section 3.6.

2.2.5 Moments of a distribution

In this section, we describe various summary statistics that can be derived from a probability distribution (either a pdf or pmf).

2.2.5.1 Mean of a distribution

The most familiar property of a distribution is its **mean**, or **expected value**, often denoted by μ. For continuous rv's, the mean is defined as follows:

$$\mathbb{E}[X] \triangleq \int_{\mathcal{X}} x\, p(x) dx \tag{2.24}$$

If the integral is not finite, the mean is not defined; we will see some examples of this later.

For discrete rv's, the mean is defined as follows:

$$\mathbb{E}[X] \triangleq \sum_{x \in \mathcal{X}} x\, p(x) \tag{2.25}$$

However, this is only meaningful if the values of x are ordered in some way (e.g., if they represent integer counts).

Since the mean is a linear operator, we have

$$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b \tag{2.26}$$

This is called the **linearity of expectation**.

For a set of n random variables, one can show that the expectation of their sum is as follows:

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] \tag{2.27}$$

If they are independent, the expectation of their product is given by

$$\mathbb{E}\left[\prod_{i=1}^{n} X_i\right] = \prod_{i=1}^{n} \mathbb{E}[X_i] \tag{2.28}$$

2.2.5.2 Variance of a distribution

The **variance** is a measure of the "spread" of a distribution, often denoted by σ^2. This is defined as follows:

$$\mathbb{V}[X] \triangleq \mathbb{E}\left[(X - \mu)^2\right] = \int (x - \mu)^2 p(x) dx \tag{2.29}$$

$$= \int x^2 p(x) dx + \mu^2 \int p(x) dx - 2\mu \int x p(x) dx = \mathbb{E}\left[X^2\right] - \mu^2 \tag{2.30}$$

from which we derive the useful result

$$\mathbb{E}\left[X^2\right] = \sigma^2 + \mu^2 \tag{2.31}$$

The **standard deviation** is defined as

$$\text{std}\left[X\right] \triangleq \sqrt{\mathbb{V}\left[X\right]} = \sigma \tag{2.32}$$

This is useful since it has the same units as X itself.

The variance of a shifted and scaled version of a random variable is given by

$$\mathbb{V}\left[aX + b\right] = a^2\mathbb{V}\left[X\right] \tag{2.33}$$

If we have a set of n independent random variables, the variance of their sum is given by the sum of their variances:

$$\mathbb{V}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{V}\left[X_i\right] \tag{2.34}$$

The variance of their product can also be derived, as follows:

$$\begin{align}
\mathbb{V}\left[\prod_{i=1}^{n} X_i\right] &= \mathbb{E}\left[(\prod_i X_i)^2\right] - (\mathbb{E}\left[\prod_i X_i\right])^2 \tag{2.35}\\
&= \mathbb{E}\left[\prod_i X_i^2\right] - (\prod_i \mathbb{E}\left[X_i\right])^2 \tag{2.36}\\
&= \prod_i \mathbb{E}\left[X_i^2\right] - \prod_i (\mathbb{E}\left[X_i\right])^2 \tag{2.37}\\
&= \prod_i (\mathbb{V}\left[X_i\right] + (\mathbb{E}\left[X_i\right])^2) - \prod_i (\mathbb{E}\left[X_i\right])^2 \tag{2.38}\\
&= \prod_i (\sigma_i^2 + \mu_i^2) - \prod_i \mu_i^2 \tag{2.39}
\end{align}$$

2.2.5.3 Mode of a distribution

The **mode** of a distribution is the value with the highest probability mass or probability density:

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}}\, p(\boldsymbol{x}) \tag{2.40}$$

If the distribution is **multimodal**, this may not be unique, as illustrated in Figure 2.4. Furthermore, even if there is a unique mode, this point may not be a good summary of the distribution.

2.2.5.4 Conditional moments

When we have two or more dependent random variables, we can compute the moments of one given knowledge of the other. For example, the **law of iterated expectations**, also called the **law of total expectation**, tells us that

$$\mathbb{E}\left[X\right] = \mathbb{E}_Y\left[\mathbb{E}\left[X|Y\right]\right] \tag{2.41}$$

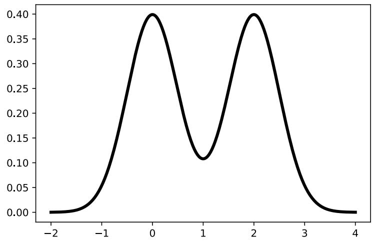


Figure 2.4: Illustration of a mixture of two 1d Gaussians, $p(x) = 0.5\mathcal{N}(x|0, 0.5) + 0.5\mathcal{N}(x|2, 0.5)$. Generated by bimodal_dist_plot.ipynb.

To prove this, let us suppose, for simplicity, that X and Y are both discrete rv's. Then we have

$$\mathbb{E}_Y\left[\mathbb{E}\left[X|Y\right]\right] = \mathbb{E}_Y\left[\sum_x x\, p(X = x|Y)\right] \tag{2.42}$$

$$= \sum_y \left[\sum_x x\, p(X = x|Y = y)\right] p(Y = y) = \sum_{x,y} x p(X = x, Y = y) = \mathbb{E}\left[X\right] \tag{2.43}$$

To give a more intuitive explanation, consider the following simple example.[3] Let X be the lifetime duration of a lightbulb, and let Y be the factory the lightbulb was produced in. Suppose $\mathbb{E}\left[X|Y = 1\right] = 5000$ and $\mathbb{E}\left[X|Y = 2\right] = 4000$, indicating that factory 1 produces longer lasting bulbs. Suppose factory 1 supplies 60% of the lightbulbs, so $p(Y = 1) = 0.6$ and $p(Y = 2) = 0.4$. Then the expected duration of a random lightbulb is given by

$$\mathbb{E}\left[X\right] = \mathbb{E}\left[X|Y = 1\right] p(Y = 1) + \mathbb{E}\left[X|Y = 2\right] p(Y = 2) = 5000 \times 0.6 + 4000 \times 0.4 = 4600 \tag{2.44}$$

There is a similar formula for the variance. In particular, the **law of total variance**, also called the **conditional variance formula**, tells us that

$$\mathbb{V}\left[X\right] = \mathbb{E}_Y\left[\mathbb{V}\left[X|Y\right]\right] + \mathbb{V}_Y\left[\mathbb{E}\left[X|Y\right]\right] \tag{2.45}$$

To see this, let us define the conditional moments, $\mu_{X|Y} = \mathbb{E}\left[X|Y\right]$, $s_{X|Y} = \mathbb{E}\left[X^2|Y\right]$, and $\sigma^2_{X|Y} = \mathbb{V}\left[X|Y\right] = s_{X|Y} - \mu^2_{X|Y}$, which are functions of Y (and therefore are random quantities). Then we have

$$\mathbb{V}\left[X\right] = \mathbb{E}\left[X^2\right] - \left(\mathbb{E}\left[X\right]\right)^2 = \mathbb{E}_Y\left[s_{X|Y}\right] - \left(\mathbb{E}_Y\left[\mu_{X|Y}\right]\right)^2 \tag{2.46}$$

$$= \mathbb{E}_Y\left[\sigma^2_{X|Y}\right] + \mathbb{E}_Y\left[\mu^2_{X|Y}\right] - \left(\mathbb{E}_Y\left[\mu_{X|Y}\right]\right)^2 \tag{2.47}$$

$$= \mathbb{E}_Y\left[\mathbb{V}\left[X|Y\right]\right] + \mathbb{V}_Y[\mu_{X|Y}] \tag{2.48}$$

To get some intuition for these formulas, consider a mixture of K univariate Gaussians. Let Y be the hidden indicator variable that specifies which mixture component we are using, and let

---

[3] This example is from `https://en.wikipedia.org/wiki/Law_of_total_expectation`, but with modified notation.

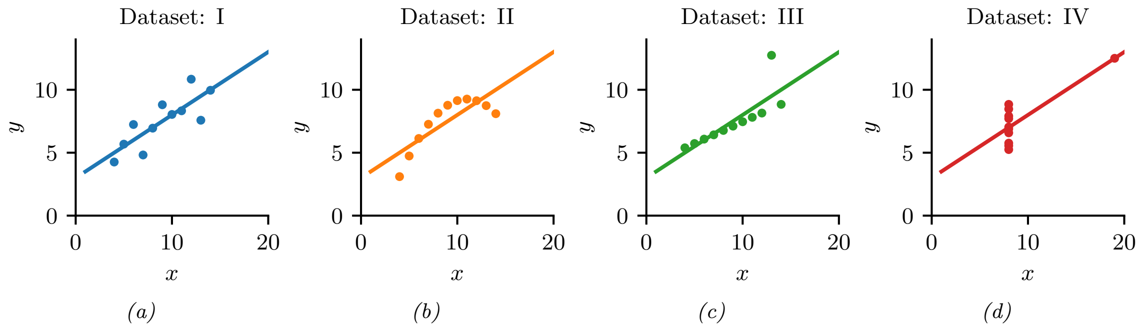


Figure 2.5: Illustration of Anscombe's quartet. All of these datasets have the same low order summary statistics. Generated by anscombes_quartet.ipynb.

$X = \sum_{y=1}^{K} \pi_y \mathcal{N}(X|\mu_y, \sigma_y)$. In Figure 2.4, we have $\pi_1 = \pi_2 = 0.5$, $\mu_1 = 0$, $\mu_2 = 2$, $\sigma_1 = \sigma_2 = 0.5$. Thus

$$\mathbb{E}\left[\mathbb{V}\left[X|Y\right]\right] = \pi_1 \sigma_1^2 + \pi_2 \sigma_2^2 = 0.25 \tag{2.49}$$

$$\mathbb{V}\left[\mathbb{E}\left[X|Y\right]\right] = \pi_1(\mu_1 - \overline{\mu})^2 + \pi_2(\mu_2 - \overline{\mu})^2 = 0.5(0-1)^2 + 0.5(2-1)^2 = 0.5 + 0.5 = 1 \tag{2.50}$$

So we get the intuitive result that the variance of X is dominated by which centroid it is drawn from (i.e., difference in the means), rather than the local variance around each centroid.

2.2.6 Limitations of summary statistics *

Although it is common to summarize a probability distribution (or points sampled from a distribution) using simple statistics such as the mean and variance, this can lose a lot of information. A striking example of this is known as **Anscombe's quartet** [Ans73], which is illustrated in Figure 2.5. This shows 4 different datasets of (x, y) pairs, all of which have identical mean, variance and correlation coefficient ρ (defined in Section 3.1.2): $\mathbb{E}[x] = 9$, $\mathbb{V}[x] = 11$, $\mathbb{E}[y] = 7.50$, $\mathbb{V}[y] = 4.12$, and $\rho = 0.816$.[4] However, the joint distributions $p(x, y)$ from which these points were sampled are clearly very different. Anscombe invented these datasets, each consisting of 10 data points, to counter the impression among statisticians that numerical summaries are superior to data visualization [Ans73].

An even more striking example of this phenomenon is shown in Figure 2.6. This consists of a dataset that looks like a dinosaur[5], plus 11 other datasets, all of which have identical low order statistics. This collection of datasets is called the **Datasaurus Dozen** [MF17]. The exact values of the (x, y) points are available online.[6] They were computed using simulated annealing, a derivative free optimization method which we discuss in the sequel to this book, [Mur23]. (The objective

---

[4] The maximum likelihood estimate for the variance in Equation (4.36) differs from the unbiased estimate in Equation (4.38). For the former, we have $\mathbb{V}[x] = 10.00$, $\mathbb{V}[y] = 3.75$, for the latter, we have $\mathbb{V}[x] = 11.00$, $\mathbb{V}[y] = 4.12$.

[5] This dataset was created by Alberto Cairo, and is available at `http://www.thefunctionalart.com/2016/08/download-datasaurus-never-trust-summary.html`

[6] `https://www.autodesk.com/research/publications/same-stats-different-graphs`. There are actually 13 datasets in total, including the dinosaur. We omitted the "away" dataset for visual clarity.

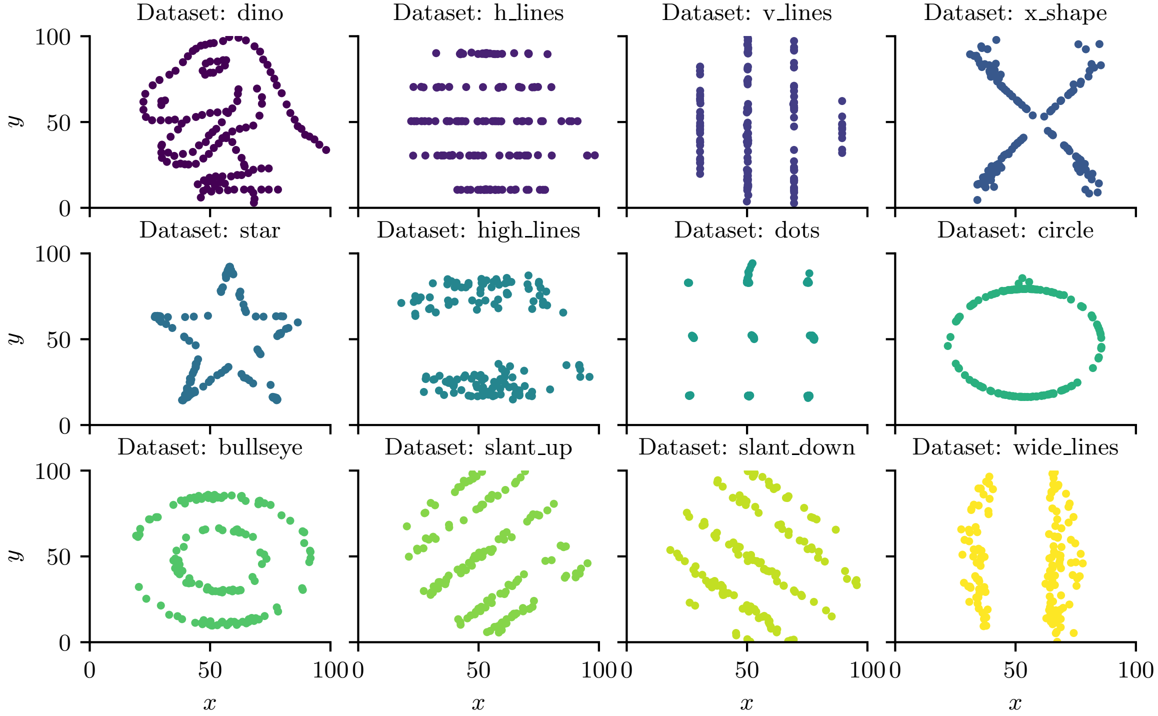


Figure 2.6: Illustration of the Datasaurus Dozen. All of these datasets have the same low order summary statistics. Adapted from Figure 1 of [MF17]. Generated by datasaurus_ dozen.ipynb.

function being optimized measures deviation from the target summary statistics of the original dinosaur, plus distance from a particular target shape.)

The same simulated annealing approach can be applied to 1d datasets, as shown in Figure 2.7. We see that all the datasets are quite different, but they all have the same median and **inter-quartile range** as shown by the central shaded part of the **box plots** in the middle. A better visualization is known as a **violin plot**, shown on the right. This shows (two copies of) the 1d kernel density estimate (Section 16.3) of the distribution on the vertical axis, in addition to the median and IQR markers. This visualization is better able to distinguish differences in the distributions. However, the technique is limited to 1d data.

2.3 Bayes' rule

> Bayes's theorem is to the theory of probability what Pythagoras's theorem is to geometry.
> — Sir Harold Jeffreys, 1973 [Jef73].

In this section, we discuss the basics of **Bayesian inference**. According to the Merriam-Webster dictionary, the term "inference" means "the act of passing from sample data to generalizations, usually with calculated degrees of certainty". The term "Bayesian" is used to refer to inference methods

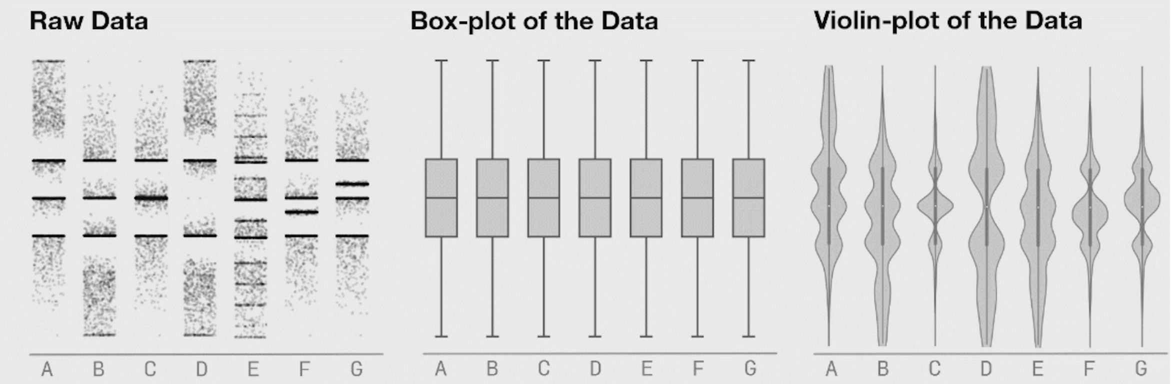


Figure 2.7: Illustration of 7 different datasets (left), the corresponding box plots (middle) and violin box plots (right). From Figure 8 of `https://www.autodesk.com/research/publications/same-stats-different-graphs`*. Used with kind permission of Justin Matejka.*

that represent "degrees of certainty" using probability theory, and which leverage **Bayes' rule**[7], to update the degree of certainty given data.

Bayes' rule itself is very simple: it is just a formula for computing the probability distribution over possible values of an unknown (or **hidden**) quantity H given some observed data $Y = y$:

$$p(H = h|Y = y) = \frac{p(H = h)p(Y = y|H = h)}{p(Y = y)} \tag{2.51}$$

This follows automatically from the identity

$$p(h|y)p(y) = p(h)p(y|h) = p(h, y) \tag{2.52}$$

which itself follows from the **product rule of probability**.

In Equation (2.51), the term $p(H)$ represents what we know about possible values of H before we see any data; this is called the **prior distribution**. (If H has K possible values, then $p(H)$ is a vector of K probabilities, that sum to 1.) The term $p(Y|H = h)$ represents the distribution over the possible outcomes Y we expect to see if $H = h$; this is called the **observation distribution**. When we evaluate this at a point corresponding to the actual observations, y, we get the function $p(Y = y|H = h)$, which is called the **likelihood**. (Note that this is a function of h, since y is fixed, but it is not a probability distribution, since it does not sum to one.) Multiplying the prior distribution $p(H = h)$ by the likelihood function $p(Y = y|H = h)$ for each h gives the unnormalized joint distribution $p(H = h, Y = y)$. We can convert this into a normalized distribution by dividing by $p(Y = y)$, which is known as the **marginal likelihood**, since it is computed by marginalizing over the unknown H:

$$p(Y = y) = \sum_{h' \in \mathcal{H}} p(H = h')p(Y = y|H = h') = \sum_{h' \in \mathcal{H}} p(H = h', Y = y) \tag{2.53}$$

---

[7] Thomas Bayes (1702–1761) was an English mathematician and Presbyterian minister. For a discussion of whether to spell this as Bayes rule, Bayes' rule or Bayes's rule, see `https://bit.ly/2kDtLuK`.

		Observation	
		0	1
Truth	0	TNR=Specificity=0.975	FPR=1-TNR=0.025
	1	FNR=1-TPR=0.125	TPR=Sensitivity=0.875

Table 2.1: Likelihood function $p(Y|H)$ for a binary observation Y given two possible hidden states H. Each row sums to one. Abbreviations: TNR is true negative rate, TPR is true positive rate, FNR is false negative rate, FPR is false positive rate.

Normalizing the joint distribution by computing $p(H = h, Y = y)/p(Y = y)$ for each h gives the **posterior distribution** $p(H = h|Y = y)$; this represents our new **belief state** about the possible values of H.

We can summarize Bayes rule in words as follows:

$$\text{posterior} \propto \text{prior} \times \text{likelihood} \tag{2.54}$$

Here we use the symbol $\propto$ to denote "proportional to", since we are ignoring the denominator, which is just a constant, independent of H. Using Bayes rule to update a distribution over unknown values of some quantity of interest, given relevant observed data, is called **Bayesian inference**, or **posterior inference**. It can also just be called **probabilistic inference**.

Below we give some simple examples of Bayesian inference in action. We will see many more interesting examples later in this book.

2.3.1 Example: Testing for COVID-19

Suppose you think you may have contracted **COVID-19**, which is an infectious disease caused by the **SARS-CoV-2** virus. You decide to take a diagnostic test, and you want to use its result to determine if you are infected or not.

Let $H = 1$ be the event that you are infected, and $H = 0$ be the event you are not infected. Let $Y = 1$ if the test is positive, and $Y = 0$ if the test is negative. We want to compute $p(H = h|Y = y)$, for $h \in \{0, 1\}$, where y is the observed test outcome. (We will write the distribution of values, $[p(H = 0|Y = y), p(H = 1|Y = y)]$ as $p(H|y)$, for brevity.) We can think of this as a form of **binary classification**, where H is the unknown class label, and y is the feature vector.

First we must specify the likelihood. This quantity obviously depends on how reliable the test is. There are two key parameters. The **sensitivity** (aka **true positive rate**) is defined as $p(Y = 1|H = 1)$, i.e., the probability of a positive test given that the truth is positive. The **false negative rate** is defined as one minus the sensitivity. The **specificity** (aka **true negative rate**) is defined as $p(Y = 0|H = 0)$, i.e., the probability of a negative test given that the truth is negative. The **false positive rate** is defined as one minus the specificity. We summarize all these quantities in Table 2.1. (See Section 5.1.3.1 for more details.) Following `https://nyti.ms/31MTZgV`, we set the sensitivity to 87.5% and the specificity to 97.5%.

Next we must specify the prior. The quantity $p(H = 1)$ represents the **prevalence** of the disease in the area in which you live. We set this to $p(H = 1) = 0.1$ (i.e., 10%), which was the prevalence in New York City in Spring 2020. (This example was chosen to match the numbers in `https://nyti.ms/31MTZgV`.)

Now suppose you test positive. We have

$$p(H=1|Y=1) = \frac{p(Y=1|H=1)p(H=1)}{p(Y=1|H=1)p(H=1)+p(Y=1|H=0)p(H=0)} \tag{2.55}$$

$$= \frac{\text{TPR} \times \text{prior}}{\text{TPR} \times \text{prior} + \text{FPR} \times (1-\text{prior})} \tag{2.56}$$

$$= \frac{0.875 \times 0.1}{0.875 \times 0.1 + 0.025 \times 0.9} = 0.795 \tag{2.57}$$

So there is a 79.5% chance you are infected.

Now suppose you test negative. The probability you are infected is given by

$$p(H=1|Y=0) = \frac{p(Y=0|H=1)p(H=1)}{p(Y=0|H=1)p(H=1)+p(Y=0|H=0)p(H=0)} \tag{2.58}$$

$$= \frac{\text{FNR} \times \text{prior}}{\text{FNR} \times \text{prior} + \text{TNR} \times (1-\text{prior})} \tag{2.59}$$

$$= \frac{0.125 \times 0.1}{0.125 \times 0.1 + 0.975 \times 0.9} = 0.014 \tag{2.60}$$

So there is just a 1.4% chance you are infected.

Nowadays COVID-19 prevalence is much lower. Suppose we repeat these calculations using a base rate of 1%; now the posteriors reduce to 26% and 0.13% respectively.

The fact that you only have a 26% chance of being infected with COVID-19, even after a positive test, is very counter-intuitive. The reason is that a single positive test is more likely to be a false positive than due to the disease, since the disease is rare. To see this, suppose we have a population of 100,000 people, of whom 1000 are infected. Of those who are infected, $875 = 0.875 \times 1000$ test positive, and of those who are uninfected, $2475 = 0.025 \times 99,000$ test positive. Thus the total number of positives is $3350 = 875 + 2475$, so the posterior probability of being infected given a positive test is $875/3350 = 0.26$.

Of course, the above calculations assume we know the sensitivity and specificity of the test. See [GC20] for how to apply Bayes rule for diagnostic testing when there is uncertainty about these parameters.

2.3.2 Example: The Monty Hall problem

In this section, we consider a more "frivolous" application of Bayes rule. In particular, we apply it to the famous **Monty Hall problem**.

Imagine a game show with the following rules: There are three doors, labeled 1, 2, 3. A single prize (e.g., a car) has been hidden behind one of them. You get to select one door. Then the gameshow host opens one of the other two doors (not the one you picked), in such a way as to not reveal the prize location. At this point, you will be given a fresh choice of door: you can either stick with your first choice, or you can switch to the other closed door. All the doors will then be opened and you will receive whatever is behind your final choice of door.

For example, suppose you choose door 1, and the gameshow host opens door 3, revealing nothing behind the door, as promised. Should you (a) stick with door 1, or (b) switch to door 2, or (c) does it make no difference?

Door 1	Door 2	Door 3	Switch	Stay
Car	-	-	Lose	Win
-	Car	-	Win	Lose
-	-	Car	Win	Lose

Table 2.2: 3 possible states for the Monty Hall game, showing that switching doors is two times better (on average) than staying with your original choice. Adapted from Table 6.1 of [PM18].

Intuitively, it seems it should make no difference, since your initial choice of door cannot influence the location of the prize. However, the fact that the host opened door 3 tells us something about the location of the prize, since he made his choice conditioned on the knowledge of the true location and on your choice. As we show below, you are in fact twice as likely to win the prize if you switch to door 2.

To show this, we will use Bayes' rule. Let H_i denote the hypothesis that the prize is behind door i. We make the following assumptions: the three hypotheses H_1, H_2 and H_3 are equiprobable *a priori*, i.e.,

$$P(H_1) = P(H_2) = P(H_3) = \frac{1}{3}. \tag{2.61}$$

The datum we receive, after choosing door 1, is either $Y = 3$ and $Y = 2$ (meaning door 3 or 2 is opened, respectively). We assume that these two possible outcomes have the following probabilities. If the prize is behind door 1, then the host selects at random between $Y = 2$ and $Y = 3$. Otherwise the choice of the host is forced and the probabilities are 0 and 1.

$$\left| \begin{array}{l} P(Y=2|H_1)=\frac{1}{2} \\ P(Y=3|H_1)=\frac{1}{2} \end{array} \right| \begin{array}{l} P(Y=2|H_2)=0 \\ P(Y=3|H_2)=1 \end{array} \left| \begin{array}{l} P(Y=2|H_3)=1 \\ P(Y=3|H_3)=0 \end{array} \right| \tag{2.62}$$

Now, using Bayes' theorem, we evaluate the posterior probabilities of the hypotheses:

$$P(H_i|Y=3) = \frac{P(Y=3|H_i)P(H_i)}{P(Y=3)} \tag{2.63}$$

$$\left| \; P(H_1|Y=3) = \tfrac{(1/2)(1/3)}{P(Y=3)} \; \right| \; P(H_2|Y=3) = \tfrac{(1)(1/3)}{P(Y=3)} \; \left| \; P(H_3|Y=3) = \tfrac{(0)(1/3)}{P(Y=3)} \; \right| \tag{2.64}$$

The denominator $P(Y = 3)$ is $P(Y = 3) = \frac{1}{6} + \frac{1}{3} = \frac{1}{2}$. So

$$\left| \; P(H_1|Y=3) \;=\; \frac{1}{3} \; \right| \; P(H_2|Y=3) \;=\; \frac{2}{3} \; \left| \; P(H_3|Y=3) \;=\; 0. \; \right| \tag{2.65}$$

So the contestant should switch to door 2 in order to have the biggest chance of getting the prize. See Table 2.2 for a worked example.

Many people find this outcome surprising. One way to make it more intuitive is to perform a thought experiment in which the game is played with a million doors. The rules are now that the contestant chooses one door, then the game show host opens 999,998 doors in such a way as not to reveal the prize, leaving the *contestant's* selected door and *one other door* closed. The contestant may now stick or switch. Imagine the contestant confronted by a million doors, of which doors 1 and 234,598 have not been opened, door 1 having been the contestant's initial guess. Where do you think the prize is?

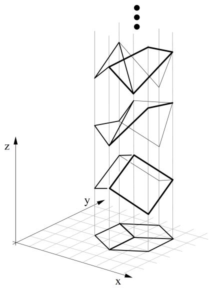


Figure 2.8: Any planar line-drawing is geometrically consistent with infinitely many 3-D structures. From Figure 11 of [SA93]. Used with kind permission of Pawan Sinha.

2.3.3 Inverse problems *

Probability theory is concerned with predicting a distribution over outcomes y given knowledge (or assumptions) about the state of the world, h. By contrast, **inverse probability** is concerned with inferring the state of the world from observations of outcomes. We can think of this as inverting the $h \to y$ mapping.

For example, consider trying to infer a 3d shape h from a 2d image y, which is a classic problem in **visual scene understanding**. Unfortunately, this is a fundamentally **ill-posed** problem, as illustrated in Figure 2.8, since there are multiple possible hidden h's consistent with the same observed y (see e.g., [Piz01]). Similarly, we can view **natural language understanding** as an ill-posed problem, in which the listener must infer the intention h from the (often ambiguous) words spoken by the speaker (see e.g., [Sab21]).

To tackle such **inverse problems**, we can use Bayes' rule to compute the posterior, $p(h|y)$, which gives a distribution over possible states of the world. This requires specifying the **forwards model**, $p(y|h)$, as well as a prior $p(h)$, which can be used to rule out (or downweight) implausible world states. We discuss this topic in more detail in the sequel to this book, [Mur23].

2.4 Bernoulli and binomial distributions

Perhaps the simplest probability distribution is the **Bernoulli distribution**, which can be used to model binary events, as we discuss below.

2.4.1 Definition

Consider tossing a coin, where the probability of event that it lands heads is given by $0 \leq \theta \leq 1$. Let $Y = 1$ denote this event, and let $Y = 0$ denote the event that the coin lands tails. Thus we are assuming that $p(Y = 1) = \theta$ and $p(Y = 0) = 1 - \theta$. This is called the **Bernoulli distribution**, and can be written as follows

$$Y \sim \mathrm{Ber}(\theta) \tag{2.66}$$

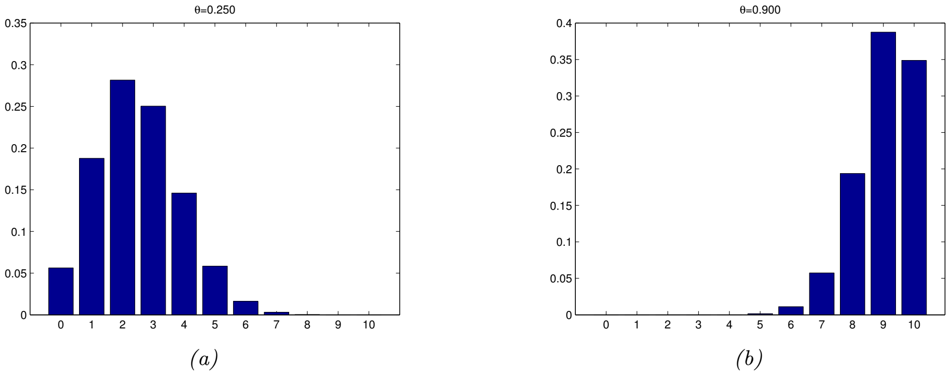


Figure 2.9: Illustration of the binomial distribution with $N = 10$ and (a) $\theta = 0.25$ and (b) $\theta = 0.9$. Generated by binom_ dist_ plot.ipynb.

where the symbol $\sim$ means "is sampled from" or "is distributed as", and Ber refers to Bernoulli. The probability mass function (pmf) of this distribution is defined as follows:

$$\text{Ber}(y|\theta) = \begin{cases} 1-\theta & \text{if } y = 0 \\ \theta & \text{if } y = 1 \end{cases} \tag{2.67}$$

(See Section 2.2.1 for details on pmf's.) We can write this in a more concise manner as follows:

$$\text{Ber}(y|\theta) \triangleq \theta^y (1-\theta)^{1-y} \tag{2.68}$$

The Bernoulli distribution is a special case of the **binomial distribution**. To explain this, suppose we observe a set of N Bernoulli trials, denoted $y_n \sim \text{Ber}(\cdot|\theta)$, for $n = 1 : N$. Concretely, think of tossing a coin N times. Let us define s to be the total number of heads, $s \triangleq \sum_{n=1}^N \mathbb{I}(y_n = 1)$. The distribution of s is given by the binomial distribution:

$$\text{Bin}(s|N,\theta) \triangleq \binom{N}{s} \theta^s (1-\theta)^{N-s} \tag{2.69}$$

where

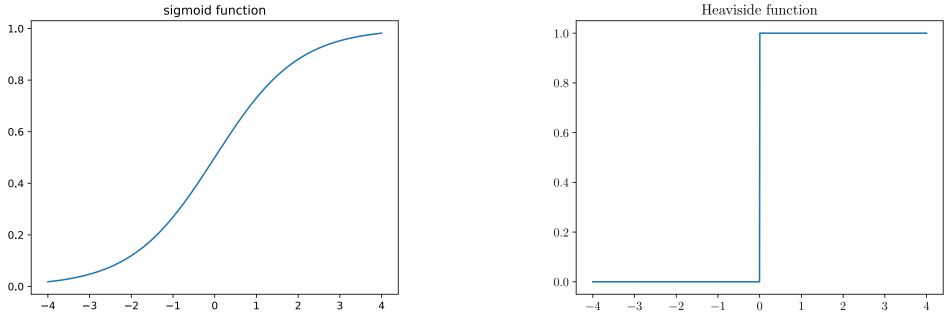
$$\binom{N}{k} \triangleq \frac{N!}{(N-k)!k!} \tag{2.70}$$

is the number of ways to choose k items from N (this is known as the **binomial coefficient**, and is pronounced "N choose k"). See Figure 2.9 for some examples of the binomial distribution. If $N = 1$, the binomial distribution reduces to the Bernoulli distribution.

2.4.2 Sigmoid (logistic) function

When we want to predict a binary variable $y \in \{0, 1\}$ given some inputs $\boldsymbol{x} \in \mathcal{X}$, we need to use a **conditional probability distribution** of the form

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \text{Ber}(y|f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{2.71}$$



(a) *(b)*

Figure 2.10: (a) The sigmoid (logistic) function $\sigma(a) = (1 + e^{-a})^{-1}$. (b) The Heaviside function $\mathbb{I}(a > 0)$. Generated by activation_fun_plot.ipynb.

$$\sigma(x) \triangleq \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{2.72}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x)) \tag{2.73}$$

$$1 - \sigma(x) = \sigma(-x) \tag{2.74}$$

$$\sigma^{-1}(p) = \log\left(\frac{p}{1-p}\right) \triangleq \text{logit}(p) \tag{2.75}$$

$$\sigma_+(x) \triangleq \log(1+e^x) \triangleq \text{softplus}(x) \tag{2.76}$$

$$\frac{d}{dx}\sigma_+(x) = \sigma(x) \tag{2.77}$$

Table 2.3: Some useful properties of the sigmoid (logistic) and related functions. Note that the **logit** *function is the inverse of the sigmoid function, and has a domain of* $[0, 1]$.

where $f(\boldsymbol{x}; \boldsymbol{\theta})$ is some function that predicts the mean parameter of the output distribution. We will consider many different kinds of function f in Part II–Part IV.

To avoid the requirement that $0 \leq f(\boldsymbol{x}; \boldsymbol{\theta}) \leq 1$, we can let f be an unconstrained function, and use the following model:

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \text{Ber}(y|\sigma(f(\boldsymbol{x}; \boldsymbol{\theta}))) \tag{2.78}$$

Here $\sigma()$ is the **sigmoid** or **logistic** function, defined as follows:

$$\sigma(a) \triangleq \frac{1}{1+e^{-a}} \tag{2.79}$$

where $a = f(\boldsymbol{x}; \boldsymbol{\theta})$. The term "sigmoid" means S-shaped: see Figure 2.10a for a plot. We see that it

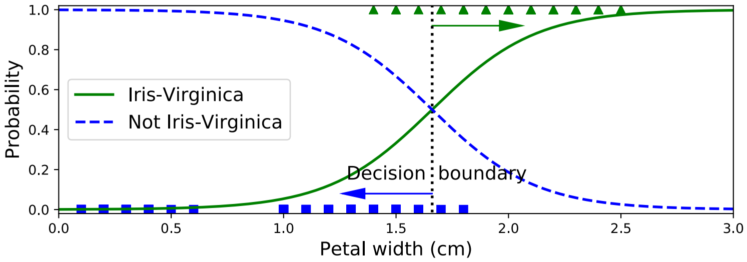


Figure 2.11: Logistic regression applied to a 1-dimensional, 2-class version of the Iris dataset. Generated by iris_logreg.ipynb. Adapted from Figure 4.23 of [Gér19].

maps the whole real line to $[0, 1]$, which is necessary for the output to be interpreted as a probability (and hence a valid value for the Bernoulli parameter θ). The sigmoid function can be thought of as a "soft" version of the **heaviside step function**, defined by

$$H(a) \triangleq \mathbb{I}(a > 0) \tag{2.80}$$

as shown in Figure 2.10b.

Plugging the definition of the sigmoid function into Equation (2.78) we get

$$p(y = 1|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{1 + e^{-a}} = \frac{e^a}{1 + e^a} = \sigma(a) \tag{2.81}$$

$$p(y = 0|\boldsymbol{x}, \boldsymbol{\theta}) = 1 - \frac{1}{1 + e^{-a}} = \frac{e^{-a}}{1 + e^{-a}} = \frac{1}{1 + e^a} = \sigma(-a) \tag{2.82}$$

The quantity a is equal to the **log odds**, $\log(\frac{p}{1-p})$, where $p = p(y = 1|\boldsymbol{x}; \boldsymbol{\theta})$. To see this, note that

$$\log\left(\frac{p}{1-p}\right) = \log\left(\frac{e^a}{1+e^a}\frac{1+e^a}{1}\right) = \log(e^a) = a \tag{2.83}$$

The **logistic function** or **sigmoid** function maps the log-odds a to p:

$$p = \text{logistic}(a) = \sigma(a) \triangleq \frac{1}{1 + e^{-a}} = \frac{e^a}{1 + e^a} \tag{2.84}$$

The inverse of this is called the **logit function**, and maps p to the log-odds a:

$$a = \text{logit}(p) = \sigma^{-1}(p) \triangleq \log\left(\frac{p}{1-p}\right) \tag{2.85}$$

See Table 2.3 for some useful properties of these functions.

2.4.3 Binary logistic regression

In this section, we use a conditional Bernoulli model, where we use a linear predictor of the form $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b$. Thus the model has the form

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \text{Ber}(y|\sigma(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b)) \tag{2.86}$$

In other words,

$$p(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} + b) = \frac{1}{1 + e^{-(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}+b)}} \tag{2.87}$$

This is called **logistic regression**.

For example consider a 1-dimensional, 2-class version of the iris dataset, where the positive class is "Virginica" and the negative class is "not Virginica", and the feature x we use is the petal width. We fit a logistic regression model to this and show the results in Figure 2.11. The **decision boundary** corresponds to the value x^* where $p(y = 1|x = x^*, \boldsymbol{\theta}) = 0.5$. We see that, in this example, $x^* \approx 1.7$. As x moves away from this boundary, the classifier becomes more confident in its prediction about the class label.

It should be clear from this example why it would be inappropriate to use linear regression for a (binary) classification problem. In such a model, the probabilities would increase above 1 as we move far enough to the right, and below 0 as we move far enough to the left.

For more detail on logistic regression, see Chapter 10.

2.5 Categorical and multinomial distributions

To represent a distribution over a finite set of labels, $y \in \{1, \ldots, C\}$, we can use the **categorical** distribution, which generalizes the Bernoulli to $C > 2$ values.

2.5.1 Definition

The categorical distribution is a discrete probability distribution with one parameter per class:

$$\mathrm{Cat}(y|\boldsymbol{\theta}) \triangleq \prod_{c=1}^{C} \theta_c^{\mathbb{I}(y=c)} \tag{2.88}$$

In other words, $p(y = c|\boldsymbol{\theta}) = \theta_c$. Note that the parameters are constrained so that $0 \leq \theta_c \leq 1$ and $\sum_{c=1}^{C} \theta_c = 1$; thus there are only $C - 1$ independent parameters.

We can write the categorical distribution in another way by converting the discrete variable y into a **one-hot vector** with C elements, all of which are 0 except for the entry corresponding to the class label. (The term "one-hot" arises from electrical engineering, where binary vectors are encoded as electrical current on a set of wires, which can be active ("hot") or not ("cold").) For example, if $C = 3$, we encode the classes 1, 2 and 3 as $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. More generally, we can encode the classes using **unit vectors**, where $\boldsymbol{e}_c$ is all 0s except for dimension c. (This is also called a **dummy encoding**.) Using one-hot encodings, we can write the categorical distribution as follows:

$$\mathrm{Cat}(\boldsymbol{y}|\boldsymbol{\theta}) \triangleq \prod_{c=1}^{C} \theta_c^{y_c} \tag{2.89}$$

The categorical distribution is a special case of the **multinomial distribution**. To explain this, suppose we observe N categorical trials, $y_n \sim \mathrm{Cat}(\cdot|\boldsymbol{\theta})$, for $n = 1 : N$. Concretely, think of rolling a C-sided dice N times. Let us define $\boldsymbol{y}$ to be a vector that counts the number of times each face

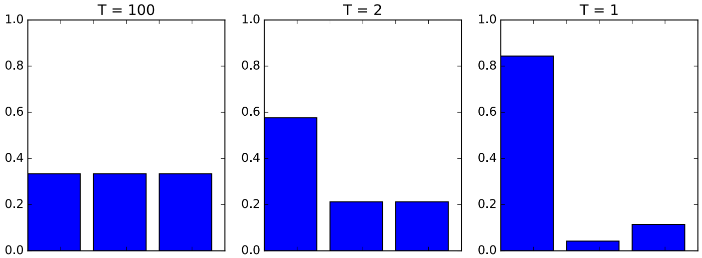


Figure 2.12: Softmax distribution $\text{softmax}(\boldsymbol{a}/T)$*, where* $\boldsymbol{a} = (3, 0, 1)$*, at temperatures of* $T = 100$*,* $T = 2$ *and* $T = 1$*. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is "spiky", with most of its mass on the largest element. Generated by softmax_plot.ipynb.*

shows up, i.e., $y_c = N_c \triangleq \sum_{n=1}^{N} \mathbb{I}(y_n = c)$. Now $\boldsymbol{y}$ is no longer one-hot, but is "multi-hot", since it has a non-zero entry for every value of c that was observed across all N trials. The distribution of $\boldsymbol{y}$ is given by the **multinomial distribution**:

$$\mathcal{M}(\boldsymbol{y}|N, \boldsymbol{\theta}) \triangleq \binom{N}{y_1 \ldots y_C} \prod_{c=1}^{C} \theta_c^{y_c} = \binom{N}{N_1 \ldots N_C} \prod_{c=1}^{C} \theta_c^{N_c} \tag{2.90}$$

where θ_c is the probability that side c shows up, and

$$\binom{N}{N_1 \ldots N_C} \triangleq \frac{N!}{N_1! N_2! \cdots N_C!} \tag{2.91}$$

is the **multinomial coefficient**, which is the number of ways to divide a set of size $N = \sum_{c=1}^{C} N_c$ into subsets with sizes N_1 up to N_C. If $N = 1$, the multinomial distribution becomes the categorical distribution.

2.5.2 Softmax function

In the conditional case, we can define

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \text{Cat}(y|f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{2.92}$$

which we can also write as

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{M}(\boldsymbol{y}|1, f(\boldsymbol{x}; \boldsymbol{\theta})) \tag{2.93}$$

We require that $0 \leq f_c(\boldsymbol{x}; \boldsymbol{\theta}) \leq 1$ and $\sum_{c=1}^{C} f_c(\boldsymbol{x}; \boldsymbol{\theta}) = 1$.

To avoid the requirement that f directly predict a probability vector, it is common to pass the output from f into the **softmax** function [Bri90], also called the **multinomial logit**. This is defined as follows:

$$\text{softmax}(\boldsymbol{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^{C} e^{a_{c'}}}, \ldots, \frac{e^{a_C}}{\sum_{c'=1}^{C} e^{a_{c'}}} \right] \tag{2.94}$$

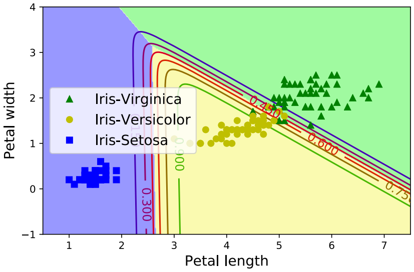


Figure 2.13: Logistic regression on the 3-class, 2-feature version of the Iris dataset. Adapted from Figure of 4.25 [Gér19]. Generated by iris_logreg.ipynb.

This maps $\mathbb{R}^C$ to $[0,1]^C$, and satisfies the constraints that $0 \leq \text{softmax}(\boldsymbol{a})_c \leq 1$ and $\sum_{c=1}^C \text{softmax}(\boldsymbol{a})_c = 1$. The inputs to the softmax, $\boldsymbol{a} = f(\boldsymbol{x};\boldsymbol{\theta})$, are called **logits**, and are a generalization of the log odds.

The softmax function is so-called since it acts a bit like the argmax function. To see this, let us divide each a_c by a constant T called the **temperature**.[8] Then as $T \to 0$, we find

$$\text{softmax}(\boldsymbol{a}/T)_c = \begin{cases} 1.0 & \text{if } c = \text{argmax}_{c'}\, a_{c'} \\ 0.0 & \text{otherwise} \end{cases} \tag{2.95}$$

In other words, at low temperatures, the distribution puts most of its probability mass in the most probable state (this is called **winner takes all**), whereas at high temperatures, it spreads the mass uniformly. See Figure 2.12 for an illustration.

2.5.3 Multiclass logistic regression

If we use a linear predictor of the form $f(\boldsymbol{x};\boldsymbol{\theta}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$, where $\mathbf{W}$ is a $C \times D$ matrix, and $\boldsymbol{b}$ is a C-dimensional bias vector, the final model becomes

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \text{Cat}(y|\text{softmax}(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})) \tag{2.96}$$

Let $\boldsymbol{a} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ be the C-dimensional vector of **logits**. Then we can rewrite the above as follows:

$$p(y = c|\boldsymbol{x};\boldsymbol{\theta}) = \frac{e^{a_c}}{\sum_{c'=1}^C e^{a_{c'}}} \tag{2.97}$$

This is known as **multinomial logistic regression**.

If we have just two classes, this reduces to binary logistic regression. To see this, note that

$$\text{softmax}(\boldsymbol{a})_0 = \frac{e^{a_0}}{e^{a_0} + e^{a_1}} = \frac{1}{1 + e^{a_1 - a_0}} = \sigma(a_0 - a_1) \tag{2.98}$$

so we can just train the model to predict $a = a_1 - a_0$. This can be done with a single weight vector $\boldsymbol{w}$; if we use the multi-class formulation, we will have two weight vectors, $\boldsymbol{w}_0$ and $\boldsymbol{w}_1$. Such a model is **over-parameterized**, which can hurt interpretability, but the predictions will be the same.

---

[8] This terminology comes from the area of statistical physics. The **Boltzmann distribution** is a distribution over states which has the same form as the softmax function.

We discuss this in more detail in Section 10.3. For now, we just give an example. Figure 2.13 shows what happens when we fit this model to the 3-class iris dataset, using just 2 features. We see that the decision boundaries between each class are linear. We can create nonlinear boundaries by transforming the features (e.g., using polynomials), as we discuss in Section 10.3.1.

2.5.4 Log-sum-exp trick

In this section, we discuss one important practical detail to pay attention to when working with the softmax distribution. Suppose we want to compute the normalized probability $p_c = p(y = c|\boldsymbol{x})$, which is given by

$$p_c = \frac{e^{a_c}}{Z(\boldsymbol{a})} = \frac{e^{a_c}}{\sum_{c'=1}^{C} e^{a_{c'}}} \tag{2.99}$$

where $\boldsymbol{a} = f(\boldsymbol{x};\boldsymbol{\theta})$ are the logits. We might encounter numerical problems when computing the **partition function** Z. For example, suppose we have 3 classes, with logits $\boldsymbol{a} = (0, 1, 0)$. Then we find $Z = e^0 + e^1 + e^0 = 4.71$. But now suppose $\boldsymbol{a} = (1000, 1001, 1000)$; we find $Z = \infty$, since on a computer, even using 64 bit precision, `np.exp(1000)=inf`. Similarly, suppose $\boldsymbol{a} = (-1000, -999, -1000)$; now we find $Z = 0$, since `np.exp(-1000)=0`. To avoid numerical problems, we can use the following identity:

$$\log \sum_{c=1}^{C} \exp(a_c) = m + \log \sum_{c=1}^{C} \exp(a_c - m) \tag{2.100}$$

This holds for any m. It is common to use $m = \max_c a_c$ which ensures that the largest value you exponentiate will be zero, so you will definitely not overflow, and even if you underflow, the answer will be sensible. This is known as the **log-sum-exp trick**. We use this trick when implementing the **lse** function:

$$\text{lse}(\boldsymbol{a}) \triangleq \log \sum_{c=1}^{C} \exp(a_c) \tag{2.101}$$

We can use this to compute the probabilities from the logits:

$$p(y = c|\boldsymbol{x}) = \exp(a_c - \text{lse}(\boldsymbol{a})) \tag{2.102}$$

We can then pass this to the cross-entropy loss, defined in Equation (5.41).

However, to save computational effort, and for numerical stability, it is quite common to modify the cross-entropy loss so that it takes the logits $\boldsymbol{a}$ as inputs, instead of the probability vector $\boldsymbol{p}$. For example, consider the binary case. The CE loss for one example is

$$\mathcal{L} = -\left[\mathbb{I}\left(y = 0\right) \log p_0 + \mathbb{I}\left(y = 1\right) \log p_1\right] \tag{2.103}$$

where

$$\log p_1 = \log\left(\frac{1}{1 + \exp(-a)}\right) = \log(1) - \log(1 + \exp(-a)) = 0 - \text{lse}([0, -a]) \tag{2.104}$$

$$\log p_0 = 0 - \text{lse}([0, +a]) \tag{2.105}$$

2.6 Univariate Gaussian (normal) distribution

The most widely used distribution of real-valued random variables $y \in \mathbb{R}$ is the **Gaussian distribution**, also called the **normal distribution** (see Section 2.6.4 for a discussion of these names).

2.6.1 Cumulative distribution function

We define the **cumulative distribution function** or **cdf** of a continuous random variable Y as follows:

$$P(y) \triangleq \Pr(Y \leq y) \tag{2.106}$$

(Note that we use a capital P to represent the cdf.) Using this, we can compute the probability of being in any interval as follows:

$$\Pr(a < Y \leq b) = P(b) - P(a) \tag{2.107}$$

Cdf's are monotonically non-decreasing functions.

The cdf of the Gaussian is defined by

$$\Phi(y; \mu, \sigma^2) \triangleq \int_{-\infty}^{y} \mathcal{N}(z|\mu, \sigma^2) dz \tag{2.108}$$

See Figure 2.2a for a plot. Note that the cdf of the Gaussian is often implemented using $\Phi(y; \mu, \sigma^2) = \frac{1}{2}[1 + \text{erf}(z/\sqrt{2})]$, where $z = (y - \mu)/\sigma$ and $\text{erf}(u)$ is the **error function**, defined as

$$\text{erf}(u) \triangleq \frac{2}{\sqrt{\pi}} \int_{0}^{u} e^{-t^2} dt \tag{2.109}$$

The parameter μ encodes the mean of the distribution; in the case of a Gaussian, this is also the same as the mode. The parameter σ^2 encodes the variance. (Sometimes we talk about the **precision** of a Gaussian, which is the inverse variance, denoted $\lambda = 1/\sigma^2$.) When $\mu = 0$ and $\sigma = 1$, the Gaussian is called the **standard normal** distribution.

If P is the cdf of Y, then $P^{-1}(q)$ is the value y_q such that $p(Y \leq y_q) = q$; this is called the q'th **quantile** of P. The value $P^{-1}(0.5)$ is the **median** of the distribution, with half of the probability mass on the left, and half on the right. The values $P^{-1}(0.25)$ and $P^{-1}(0.75)$ are the lower and upper **quartiles**.

For example, let Φ be the cdf of the Gaussian distribution $\mathcal{N}(0, 1)$, and Φ^{-1} be the inverse cdf (also known as the **probit function**). Then points to the left of $\Phi^{-1}(\alpha/2)$ contain $\alpha/2$ of the probability mass, as illustrated in Figure 2.2b. By symmetry, points to the right of $\Phi^{-1}(1 - \alpha/2)$ also contain $\alpha/2$ of the mass. Hence the central interval $(\Phi^{-1}(\alpha/2), \Phi^{-1}(1 - \alpha/2))$ contains $1 - \alpha$ of the mass. If we set $\alpha = 0.05$, the central 95% interval is covered by the range

$$(\Phi^{-1}(0.025), \Phi^{-1}(0.975)) = (-1.96, 1.96) \tag{2.110}$$

If the distribution is $\mathcal{N}(\mu, \sigma^2)$, then the 95% interval becomes $(\mu - 1.96\sigma, \mu + 1.96\sigma)$. This is often approximated by writing $\mu \pm 2\sigma$.

2.6.2 Probability density function

We define the **probability density function** or **pdf** as the derivative of the cdf:

$$p(y) \triangleq \frac{d}{dy} P(y) \tag{2.111}$$

The pdf of the Gaussian is given by

$$\mathcal{N}(y|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2} \tag{2.112}$$

where $\sqrt{2\pi\sigma^2}$ is the normalization constant needed to ensure the density integrates to 1 (see Exercise 2.12). See Figure 2.2b for a plot.

Given a pdf, we can compute the probability of a continuous variable being in a finite interval as follows:

$$\Pr(a < Y \leq b) = \int_a^b p(y) dy = P(b) - P(a) \tag{2.113}$$

As the size of the interval gets smaller, we can write

$$\Pr(y \leq Y \leq y + dy) \approx p(y) dy \tag{2.114}$$

Intuitively, this says the probability of Y being in a small interval around y is the density at y times the width of the interval. One important consequence of the above result is that the pdf at a point can be larger than 1. For example, $\mathcal{N}(0|0, 0.1) = 3.99$.

We can use the pdf to compute the **mean**, or **expected value**, of the distribution:

$$\mathbb{E}[Y] \triangleq \int_{\mathcal{Y}} y \, p(y) dy \tag{2.115}$$

For a Gaussian, we have the familiar result that $\mathbb{E}\left[\mathcal{N}(\cdot|\mu, \sigma^2)\right] = \mu$. (Note, however, that for some distributions, this integral is not finite, so the mean is not defined.)

We can also use the pdf to compute the **variance** of a distribution. This is a measure of the "spread", and is often denoted by σ^2. The variance is defined as follows:

$$\mathbb{V}[Y] \triangleq \mathbb{E}\left[(Y - \mu)^2\right] = \int (y - \mu)^2 p(y) dy \tag{2.116}$$

$$= \int y^2 p(y) dy + \mu^2 \int p(y) dy - 2\mu \int y p(y) dy = \mathbb{E}\left[Y^2\right] - \mu^2 \tag{2.117}$$

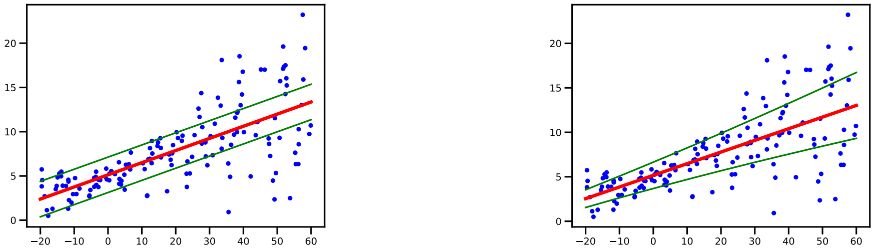
from which we derive the useful result

$$\mathbb{E}\left[Y^2\right] = \sigma^2 + \mu^2 \tag{2.118}$$

The **standard deviation** is defined as

$$\mathrm{std}[Y] \triangleq \sqrt{\mathbb{V}[Y]} = \sigma \tag{2.119}$$

(The standard deviation can be more intepretable than the variance since it has the same units as Y itself.) For a Gaussian, we have the familiar result that $\mathrm{std}\left[\mathcal{N}(\cdot|\mu, \sigma^2)\right] = \sigma$.



(a) (b)

Figure 2.14: Linear regression using Gaussian output with mean $\mu(x) = b + wx$ and (a) fixed variance σ^2 (homoskedastic) or (b) input-dependent variance $\sigma(x)^2$ (heteroscedastic). Generated by linreg_1d_hetero_tfp.ipynb.

2.6.3 Regression

So far we have been considering the unconditional Gaussian distribution. In some cases, it is helpful to make the parameters of the Gaussian be functions of some input variables, i.e., we want to create a conditional density model of the form

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(y|f_\mu(\boldsymbol{x};\boldsymbol{\theta}), f_\sigma(\boldsymbol{x};\boldsymbol{\theta})^2) \tag{2.120}$$

where $f_\mu(\boldsymbol{x};\boldsymbol{\theta}) \in \mathbb{R}$ predicts the mean, and $f_\sigma(\boldsymbol{x};\boldsymbol{\theta})^2 \in \mathbb{R}_+$ predicts the variance.

It is common to assume that the variance is fixed, and is independent of the input. This is called **homoscedastic regression**. Furthermore it is common to assume the mean is a linear function of the input. The resulting model is called **linear regression**:

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b, \sigma^2) \tag{2.121}$$

where $\boldsymbol{\theta} = (\boldsymbol{w}, b, \sigma^2)$. See Figure 2.14(a) for an illustration of this model in 1d. and Section 11.2 for more details on this model.

However, we can also make the variance depend on the input; this is called **heteroskedastic regression**. In the linear regression setting, we have

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{w}_\mu^\mathsf{T}\boldsymbol{x} + b, \sigma_+(\boldsymbol{w}_\sigma^\mathsf{T}\boldsymbol{x})) \tag{2.122}$$

where $\boldsymbol{\theta} = (\boldsymbol{w}_\mu, \boldsymbol{w}_\sigma)$ are the two forms of regression weights, and

$$\sigma_+(a) = \log(1 + e^a) \tag{2.123}$$

is the **softplus** function, that maps from $\mathbb{R}$ to $\mathbb{R}_+$, to ensure the predicted standard deviation is non-negative. See Figure 2.14(b) for an illustration of this model in 1d.

Note that Figure 2.14 plots the 95% predictive interval, $[\mu(x) - 2\sigma(x), \mu(x) + 2\sigma(x)]$. This is the uncertainty in the predicted *observation* y given $\boldsymbol{x}$, and captures the variability in the blue dots. By contrast, the uncertainty in the underlying (noise-free) function is represented by $\sqrt{\mathbb{V}[f_\mu(\boldsymbol{x};\boldsymbol{\theta})]}$, which does not involve the σ term; now the uncertainty is over the parameters $\boldsymbol{\theta}$, rather than the output y. See Section 11.7 for details on how to model parameter uncertainty.

2.6.4 Why is the Gaussian distribution so widely used?

The Gaussian distribution is the most widely used distribution in statistics and machine learning. There are several reasons for this. First, it has two parameters which are easy to interpret, and which capture some of the most basic properties of a distribution, namely its mean and variance. Second, the central limit theorem (Section 2.8.6) tells us that sums of independent random variables have an approximately Gaussian distribution, making it a good choice for modeling residual errors or "noise". Third, the Gaussian distribution makes the least number of assumptions (has maximum entropy), subject to the constraint of having a specified mean and variance, as we show in Section 3.4.4; this makes it a good default choice in many cases. Finally, it has a simple mathematical form, which results in easy to implement, but often highly effective, methods, as we will see in Section 3.2.

From a historical perspective, it's worth remarking that the term "Gaussian distribution" is a bit misleading, since, as Jaynes [Jay03, p241] notes: "The fundamental nature of this distribution and its main properties were noted by Laplace when Gauss was six years old; and the distribution itself had been found by de Moivre before Laplace was born". However, Gauss popularized the use of the distribution in the 1800s, and the term "Gaussian" is now widely used in science and engineering.

The name "normal distribution" seems to have arisen in connection with the normal equations in linear regression (see Section 11.2.2.2). However, we prefer to avoid the term "normal", since it suggests other distributions are "abnormal", whereas, as Jaynes [Jay03] points out, it is the Gaussian that is abnormal in the sense that it has many special properties that are untypical of general distributions.

2.6.5 Dirac delta function as a limiting case

As the variance of a Gaussian goes to 0, the distribution approaches an infinitely narrow, but infinitely tall, "spike" at the mean. We can write this as follows:

$$\lim_{\sigma \to 0} \mathcal{N}(y|\mu, \sigma^2) \to \delta(y - \mu) \tag{2.124}$$

where δ is the **Dirac delta function**, defined by

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases} \tag{2.125}$$

where

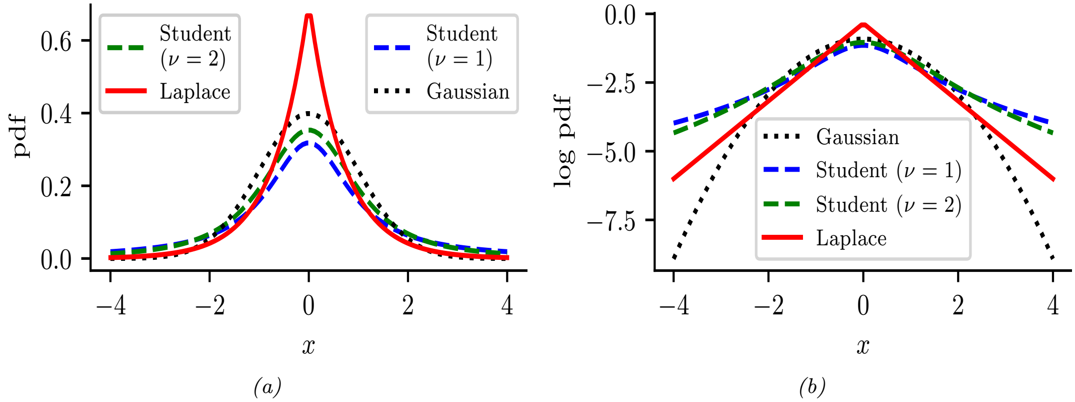
$$\int_{-\infty}^{\infty} \delta(x) dx = 1 \tag{2.126}$$

A slight variant of this is to define

$$\delta_y(x) = \begin{cases} +\infty & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases} \tag{2.127}$$

Note that we have

$$\delta_y(x) = \delta(x - y) \tag{2.128}$$



(a)

(b)

Figure 2.15: (a) The pdf's for a $\mathcal{N}(0,1)$*,* $\mathcal{T}(\mu=0,\sigma=1,\nu=1)$*,* $\mathcal{T}(\mu=0,\sigma=1,\nu=2)$*, and* $\text{Laplace}(0,1/\sqrt{2})$*. The mean is 0 and the variance is 1 for both the Gaussian and Laplace. When* $\nu=1$*, the Student is the same as the Cauchy, which does not have a well-defined mean and variance. (b) Log of these pdf's. Note that the Student distribution is not log-concave for any parameter value, unlike the Laplace distribution. Nevertheless, both are unimodal. Generated by student_laplace_pdf_plot.ipynb.*

The delta function distribution satisfies the following **sifting property**, which we will use later on:

$$\int_{-\infty}^{\infty} f(y)\delta(x-y)dy = f(x) \tag{2.129}$$

2.7 Some other common univariate distributions *

In this section, we briefly introduce some other univariate distributions that we will use in this book.

2.7.1 Student *t* distribution

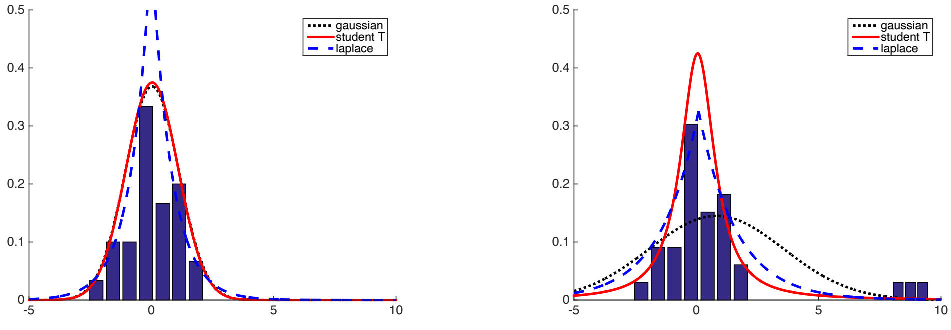
The Gaussian distribution is quite sensitive to **outliers**. A **robust** alternative to the Gaussian is the **Student *t*-distribution**, which we shall call the **Student distribution** for short.[9] Its pdf is as follows:

$$\mathcal{T}(y|\mu,\sigma^2,\nu) \propto \left[1+\frac{1}{\nu}\left(\frac{y-\mu}{\sigma}\right)^2\right]^{-\left(\frac{\nu+1}{2}\right)} \tag{2.130}$$

where μ is the mean, $\sigma > 0$ is the scale parameter (not the standard deviation), and $\nu > 0$ is called the **degrees of freedom** (although a better term would be the **degree of normality** [Kru13], since large values of ν make the distribution act like a Gaussian).

---

[9] This distribution has a colorful etymology. It was first published in 1908 by William Sealy Gosset, who worked at the Guinness brewery in Dublin, Ireland. Since his employer would not allow him to use his own name, he called it the "Student" distribution. The origin of the term t seems to have arisen in the context of tables of the Student distribution, used by Fisher when developing the basis of classical statistical inference. See `http://jeff560.tripod.com/s.html` for more historical details.



(a) (b)

Figure 2.16: Illustration of the effect of outliers on fitting Gaussian, Student and Laplace distributions. (a) No outliers (the Gaussian and Student curves are on top of each other). (b) With outliers. We see that the Gaussian is more affected by outliers than the Student and Laplace distributions. Adapted from Figure 2.16 of [Bis06]. Generated by robust_pdf_plot.ipynb.

We see that the probability density decays as a polynomial function of the squared distance from the center, as opposed to an exponential function, so there is more probability mass in the tail than with a Gaussian distribution, as shown in Figure 2.15. We say that the Student distribution has **heavy tails**, which makes it robust to outliers.

To illustrate the robustness of the Student distribution, consider Figure 2.16. On the left, we show a Gaussian and a Student distribution fit to some data with no outliers. On the right, we add some outliers. We see that the Gaussian is affected a lot, whereas the Student hardly changes. We discuss how to use the Student distribution for robust linear regression in Section 11.6.2.

For later reference, we note that the Student distribution has the following properties:

$$\text{mean} = \mu, \ \text{mode} = \mu, \ \text{var} = \frac{\nu\sigma^2}{(\nu - 2)} \tag{2.131}$$

The mean is only defined if $\nu > 1$. The variance is only defined if $\nu > 2$. For $\nu \gg 5$, the Student distribution rapidly approaches a Gaussian distribution and loses its robustness properties. It is common to use $\nu = 4$, which gives good performance in a range of problems [LLT89].

2.7.2 Cauchy distribution

If $\nu = 1$, the Student distribution is known as the **Cauchy** or **Lorentz** distribution. Its pdf is defined by

$$\mathcal{C}(x|\mu, \gamma) = \frac{1}{\gamma\pi}\left[1 + \left(\frac{x-\mu}{\gamma}\right)^2\right]^{-1} \tag{2.132}$$

This distribution has very heavy tails compared to a Gaussian. For example, 95% of the values from a standard normal are between -1.96 and 1.96, but for a standard Cauchy they are between -12.7 and 12.7. In fact the tails are so heavy that the integral that defines the mean does not converge.

The **half Cauchy** distribution is a version of the Cauchy (with $\mu = 0$) that is "folded over" on itself, so all its probability density is on the positive reals. Thus it has the form

$$\mathcal{C}_+(x|\gamma) \triangleq \frac{2}{\pi\gamma}\left[1 + \left(\frac{x}{\gamma}\right)^2\right]^{-1} \tag{2.133}$$

This is useful in Bayesian modeling, where we want to use a distribution over positive reals with heavy tails, but finite density at the origin.

2.7.3 Laplace distribution

Another distribution with heavy tails is the **Laplace distribution**[10], also known as the **double sided exponential** distribution. This has the following pdf:

$$\text{Laplace}(y|\mu, b) \triangleq \frac{1}{2b}\exp\left(-\frac{|y-\mu|}{b}\right) \tag{2.134}$$

See Figure 2.15 for a plot. Here μ is a location parameter and $b > 0$ is a scale parameter. This distribution has the following properties:

$$\text{mean} = \mu, \ \text{mode} = \mu, \ \text{var} = 2b^2 \tag{2.135}$$

In Section 11.6.1, we discuss how to use the Laplace distribution for robust linear regression, and in Section 11.4, we discuss how to use the Laplace distribution for sparse linear regression.

2.7.4 Beta distribution

The **beta distribution** has support over the interval $[0, 1]$ and is defined as follows:

$$\text{Beta}(x|a, b) = \frac{1}{B(a,b)}x^{a-1}(1-x)^{b-1} \tag{2.136}$$

where $B(a, b)$ is the **beta function**, defined by

$$B(a,b) \triangleq \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{2.137}$$

where $\Gamma(a)$ is the Gamma function defined by

$$\Gamma(a) \triangleq \int_0^\infty x^{a-1}e^{-x}dx \tag{2.138}$$

See Figure 2.17a for plots of some beta distributions.

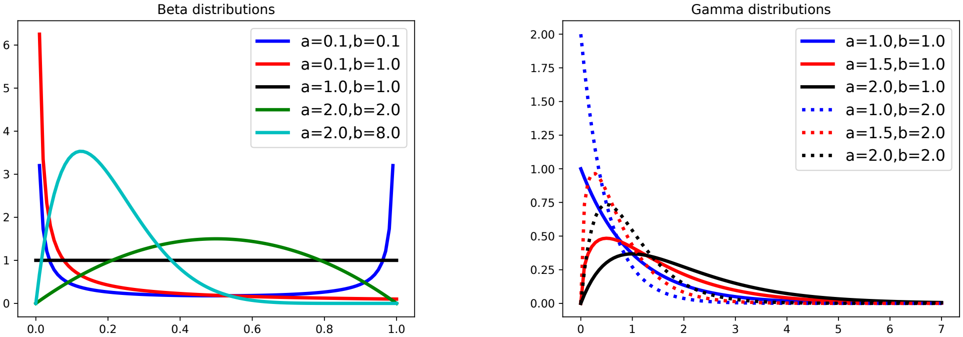
We require $a, b > 0$ to ensure the distribution is integrable (i.e., to ensure $B(a, b)$ exists). If $a = b = 1$, we get the uniform distribution. If a and b are both less than 1, we get a bimodal distribution with "spikes" at 0 and 1; if a and b are both greater than 1, the distribution is unimodal.

For later reference, we note that the distribution has the following properties (Exercise 2.8):

$$\text{mean} = \frac{a}{a+b}, \ \text{mode} = \frac{a-1}{a+b-2}, \ \text{var} = \frac{ab}{(a+b)^2(a+b+1)} \tag{2.139}$$

---

[10] Pierre-Simon Laplace (1749–1827) was a French mathematician, who played a key role in creating the field of Bayesian statistics.



(a) *(b)*

Figure 2.17: (a) Some beta distributions. If $a < 1$, we get a "spike" on the left, and if $b < 1$, we get a "spike" on the right. if $a = b = 1$, the distribution is uniform. If $a > 1$ and $b > 1$, the distribution is unimodal. Generated by beta_dist_plot.ipynb. (b) Some gamma distributions. If $a \leq 1$, the mode is at 0, otherwise the mode is away from 0. As we increase the rate b, we reduce the horizontal scale, thus squeezing everything leftwards and upwards. Generated by gamma_dist_plot.ipynb.

2.7.5 Gamma distribution

The **gamma distribution** is a flexible distribution for positive real valued rv's, $x > 0$. It is defined in terms of two parameters, called the shape $a > 0$ and the rate $b > 0$:

$$\mathrm{Ga}(x|\mathrm{shape} = a, \mathrm{rate} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{a-1} e^{-xb} \tag{2.140}$$

Sometimes the distribution is parameterized in terms of the shape a and the **scale** $s = 1/b$:

$$\mathrm{Ga}(x|\mathrm{shape} = a, \mathrm{scale} = s) \triangleq \frac{1}{s^a \Gamma(a)} x^{a-1} e^{-x/s} \tag{2.141}$$

See Figure 2.17b for some plots of the gamma pdf.

For reference, we note that the distribution has the following properties:

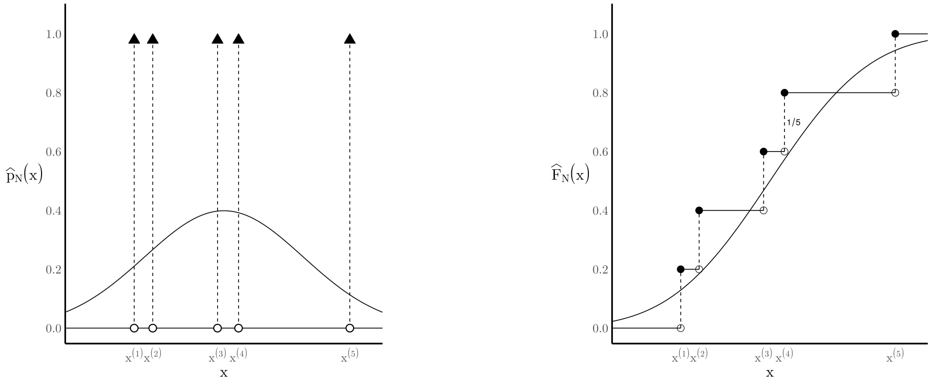
$$\mathrm{mean} = \frac{a}{b}, \ \mathrm{mode} = \frac{a-1}{b}, \ \mathrm{var} = \frac{a}{b^2} \tag{2.142}$$

There are several distributions which are just special cases of the Gamma, which we discuss below.

- **Exponential distribution**. This is defined by

  $$\mathrm{Expon}(x|\lambda) \triangleq \mathrm{Ga}(x|\mathrm{shape} = 1, \mathrm{rate} = \lambda) \tag{2.143}$$

  This distribution describes the times between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate λ.



(a) *(b)*

Figure 2.18: Illustration of the (a) empirical pdf and (b) empirical cdf derived from a set of $N = 5$ samples. From https://bit.ly/3hFgi0e*. Used with kind permission of Mauro Escudero.*

- **Chi-squared distribution**. This is defined by

  $$\chi^2_\nu(x) \triangleq \text{Ga}(x|\text{shape} = \frac{\nu}{2}, \text{rate} = \frac{1}{2}) \tag{2.144}$$

  where ν is called the degrees of freedom. This is the distribution of the sum of squared Gaussian random variables. More precisely, if $Z_i \sim \mathcal{N}(0, 1)$, and $S = \sum_{i=1}^{\nu} Z_i^2$, then $S \sim \chi^2_\nu$.

- The **inverse Gamma distribution** is defined as follows:

  $$\text{IG}(x|\text{shape} = a, \text{scale} = b) \triangleq \frac{b^a}{\Gamma(a)} x^{-(a+1)} e^{-b/x} \tag{2.145}$$

  The distribution has these properties

  $$\text{mean} = \frac{b}{a-1}, \text{mode} = \frac{b}{a+1}, \text{var} = \frac{b^2}{(a-1)^2(a-2)} \tag{2.146}$$

  The mean only exists if $a > 1$. The variance only exists if $a > 2$. Note: if $X \sim \text{Ga}(\text{shape} = a, \text{rate} = b)$, then $1/X \sim \text{IG}(\text{shape} = a, \text{scale} = b)$. (Note that b plays two different roles in this case.)

2.7.6 Empirical distribution

Suppose we have a set of N samples $\mathcal{D} = \{x^{(1)}, \ldots, x^{(N)}\}$, derived from a distribution $p(X)$, where $X \in \mathbb{R}$. We can approximate the pdf using a set of delta functions (Section 2.6.5) or "spikes", centered on these samples:

$$\hat{p}_N(x) = \frac{1}{N} \sum_{n=1}^{N} \delta_{x^{(n)}}(x) \tag{2.147}$$

This is called the **empirical distribution** of the dataset $\mathcal{D}$. An example of this, with $N = 5$, is shown in Figure 2.18(a).

The corresponding cdf is given by

$$\hat{P}_N(x) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}\left(x^{(n)} \leq x\right) = \frac{1}{N}\sum_{n=1}^{N} u_{x^{(n)}}(x) \tag{2.148}$$

where $u_y(x)$ is a **step function** at y defined by

$$u_y(x) = \begin{cases} 1 & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases} \tag{2.149}$$

This can be visualized as a "stair case", as in Figure 2.18(b), where the jumps of height $1/N$ occur at every sample.

2.8 Transformations of random variables *

Suppose $\boldsymbol{x} \sim p()$ is some random variable, and $\boldsymbol{y} = f(\boldsymbol{x})$ is some deterministic transformation of it. In this section, we discuss how to compute $p(\boldsymbol{y})$.

2.8.1 Discrete case

If X is a discrete rv, we can derive the pmf for Y by simply summing up the probability mass for all the x's such that $f(x) = y$:

$$p_y(y) = \sum_{x: f(x) = y} p_x(x) \tag{2.150}$$

For example, if $f(X) = 1$ if X is even and $f(X) = 0$ otherwise, and $p_x(X)$ is uniform on the set $\{1, \ldots, 10\}$, then $p_y(1) = \sum_{x \in \{2,4,6,8,10\}} p_x(x) = 0.5$, and hence $p_y(0) = 0.5$ also. Note that in this example, f is a many-to-one function.

2.8.2 Continuous case

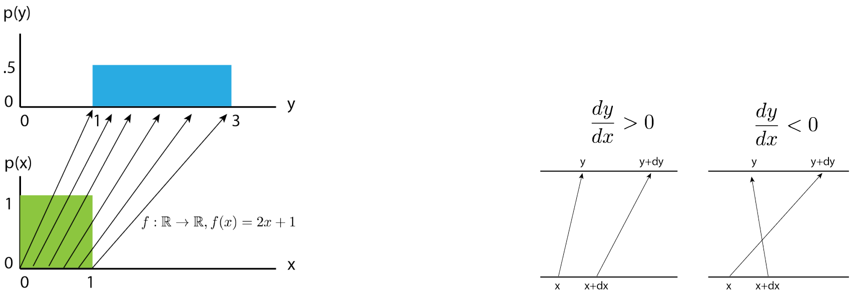
If X is continuous, we cannot use Equation (2.150) since $p_x(x)$ is a density, not a pmf, and we cannot sum up densities. Instead, we work with cdf's, as follows:

$$P_y(y) \triangleq \Pr(Y \leq y) = \Pr(f(X) \leq y) = \Pr(X \in \{x | f(x) \leq y\}) \tag{2.151}$$

If f is invertible, we can derive the pdf of y by differentiating the cdf, as we show below. If f is not invertible, we can use numerical integration, or a Monte Carlo approximation.

2.8.3 Invertible transformations (bijections)

In this section, we consider the case of monotonic and hence invertible functions. (Note a function is invertible iff it is a **bijector**). With this assumption, there is a simple formula for the pdf of y, as we will see. (This can be generalized to invertible, but non-monotonic, functions, but we ignore this case.)



(a) (b)

Figure 2.19: (a) Mapping a uniform pdf through the function $f(x) = 2x + 1$. (b) Illustration of how two nearby points, x and $x + dx$, get mapped under f. If $\frac{dy}{dx} > 0$, the function is locally increasing, but if $\frac{dy}{dx} < 0$, the function is locally decreasing. (In the latter case, if $f(x) = y + dy$, then $f(x + dx) = y$, since increasing x by dx should decrease the output by dy.) $x + dx > x$. From [Jan18]. Used with kind permission of Eric Jang.

2.8.3.1 Change of variables: scalar case

We start with an example. Suppose $x \sim \text{Unif}(0, 1)$, and $y = f(x) = 2x + 1$. This function stretches and shifts the probability distribution, as shown in Figure 2.19(a). Now let us zoom in on a point x and another point that is infinitesimally close, namely $x + dx$. We see this interval gets mapped to $(y, y + dy)$. The probability mass in these intervals must be the same, hence $p(x)dx = p(y)dy$, and so $p(y) = p(x)dx/dy$. However, since it does not matter (in terms of probability preservation) whether $dx/dy > 0$ or $dx/dy < 0$, we get

$$p_y(y) = p_x(x)|\frac{dx}{dy}| \tag{2.152}$$

Now consider the general case for any $p_x(x)$ and any monotonic function $f : \mathbb{R} \to \mathbb{R}$. Let $g = f^{-1}$, so $y = f(x)$ and $x = g(y)$. If we assume that $f : \mathbb{R} \to \mathbb{R}$ is monotonically increasing we get

$$P_y(y) = \Pr(f(X) \leq y) = \Pr(X \leq f^{-1}(y)) = P_x(f^{-1}(y)) = P_x(g(y)) \tag{2.153}$$

Taking derivatives we get

$$p_y(y) \triangleq \frac{d}{dy} P_y(y) = \frac{d}{dy} P_x(x) = \frac{dx}{dy}\frac{d}{dx} P_x(x) = \frac{dx}{dy} p_x(x) \tag{2.154}$$

We can derive a similar expression (but with opposite signs) for the case where f is monotonically decreasing. To handle the general case we take the absolute value to get

$$p_y(y) = p_x\left(g(y)\right)|\frac{d}{dy} g(y)| \tag{2.155}$$

This is called **change of variables** formula.

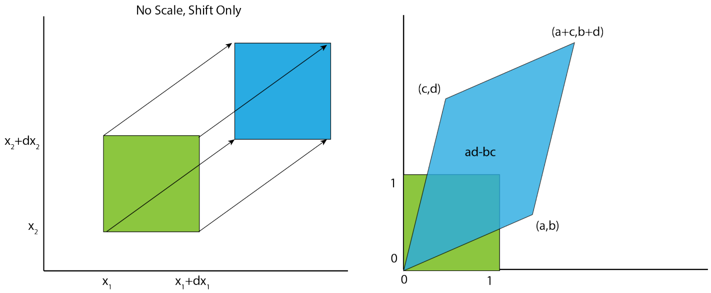


Figure 2.20: Illustration of an affine transformation applied to a unit square, $f(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} + \boldsymbol{b}$. (a) Here $\mathbf{A} = \mathbf{I}$. (b) Here $\boldsymbol{b} = \mathbf{0}$. From [Jan18]. Used with kind permission of Eric Jang.

2.8.3.2 Change of variables: multivariate case

We can extend the previous results to multivariate distributions as follows. Let $\boldsymbol{f}$ be an invertible function that maps $\mathbb{R}^n$ to $\mathbb{R}^n$, with inverse $\boldsymbol{g}$. Suppose we want to compute the pdf of $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$. By analogy with the scalar case, we have

$$p_y(\boldsymbol{y}) = p_x\left(\boldsymbol{g}(\boldsymbol{y})\right) \Big| \det\left[\mathbf{J}_g(\boldsymbol{y})\right] \Big| \tag{2.156}$$

where $\mathbf{J}_g = \frac{d\boldsymbol{g}(\boldsymbol{y})}{d\boldsymbol{y}^\mathsf{T}}$ is the Jacobian of $\boldsymbol{g}$, and $|\det \mathbf{J}(\boldsymbol{y})|$ is the absolute value of the determinant of $\mathbf{J}$ evaluated at $\boldsymbol{y}$. (See Section 7.8.5 for a discussion of Jacobians.) In Exercise 3.6 you will use this formula to derive the normalization constant for a multivariate Gaussian.

Figure 2.20 illustrates this result in 2d, for the case where $f(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} + \boldsymbol{b}$, where $\mathbf{A} = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$. We see that the area of the unit square changes by a factor of $\det(\mathbf{A}) = ad - bc$, which is the area of the parallelogram.

As another example, consider transforming a density from Cartesian coordinates $\boldsymbol{x} = (x_1, x_2)$ to polar coordinates $\boldsymbol{y} = \boldsymbol{f}(x_1, x_2)$, so $\boldsymbol{g}(r, \theta) = (r\cos\theta, r\sin\theta)$. Then

$$\mathbf{J}_g = \begin{pmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix} \tag{2.157}$$

$$|\det(\mathbf{J}_g)| = |r\cos^2\theta + r\sin^2\theta| = |r| \tag{2.158}$$

Hence

$$p_{r,\theta}(r, \theta) = p_{x_1,x_2}(r\cos\theta, r\sin\theta)\, r \tag{2.159}$$

To see this geometrically, notice that the area of the shaded patch in Figure 2.21 is given by

$$\Pr(r \le R \le r + dr, \theta \le \Theta \le \theta + d\theta) = p_{r,\theta}(r, \theta) dr d\theta \tag{2.160}$$

In the limit, this is equal to the density at the center of the patch times the size of the patch, which is given by $r\ dr\ d\theta$. Hence

$$p_{r,\theta}(r, \theta)\ dr\ d\theta = p_{x_1,x_2}(r\cos\theta, r\sin\theta)\ r\ dr\ d\theta \tag{2.161}$$