Chapter 2: Decision Trees and Random Forest (pp3-36)

p4: Data science workflow
p5: Regression, classification, decision trees, random forests
p6: Decision tree example
p11: Decision tree
pp.12-14: Precison, recall, accuracy
p15: Issues with trees
p16: Finding good splitting condition: gini purity and entropy
p17: Algorithm to learn decision tree
pp18, 19: Example: splitting conditions
p20: Random forests
p21: Decision trees for regression
pp23-36: Decision trees in Spark's MLlib

Entropy as impurity measure: In the context of decision trees, entropy is used as a criterion to decide the best split at each node. Entropy here measures the impurity of a dataset before and after a split. If a dataset is completely pure (i.e., all data points belong to the same class), the entropy is 0. If the data is evenly distributed among different classes, the entropy is high.

Information gain: The decision tree algorithm aims to reduce entropy or impurity at each split. It calculates the information gain, which is the difference between the entropy of the parent node and the weighted sum of the entropies of the child nodes. The split that maximizes information gain is chosen as the best split.

Chapter 3: Recommender Systems via Matrix Factorization (pp. 37-69)

p38: Recommender system definition
p42: Matrix factorization
pp43-44: Alternating least squares (ALS)
p45: Distributed ALS: method 1
p46: Distributed ALS: method 2
p47: Distributed ALS: runtimes
pp48-68: Recommender Systems in Spark's MLlib
pp49-52: Preparing dataset
p53: Broadcasting closure variables
p54: Training data
p55: Training recommender system
pp58-61: Evaluating model: ROC, AUC, TRP (sensitiviy), FPR (specificity)
pp62-63: Cross-validation
p64: Baseline selection
pp65-66: Hyperparameters tuning
p67: Creating batch recommentations

Chapter 4: Text processing with latent semantic analysis (pp69-108)

p70: How to analyze text data (+TF-IDF)
p73: Latent semantic (LS) analysis vs LS indexing
p74: SVD
pp75-77: SVD + cosine similarity
pp78-80: SVD reduced decomposition
p81: Processing queries
pp82-83: Finding SVD of a word-document matrix
pp84-: Text analysis and SVD in Spark's MLlib
p87: NLP pipeline
p88: Filtering
p89: From text to word lemmas
p91: Computing TF weights
pp93-95: Computing DF weights
p96: Broadcasting term dictionary
p97: Merging TF & IDF weights into sparse vectors
p98 : Computing SVD
p99: Finding top terms for latent concepts
p100: Finding top documents for latent concepts
p103: Finding most similar documents to a given query document
p104: Finding top similar terms to a given query term
p105: Finding top similar documents to a given query term
pp106-107: Multi-term queries
p108: Summary

Chapter 5: K-Means Clustering & Anomaly Detection (pp109-141)

p110: Clustering
p114: Clustering objective
p115: Naive clustering algorithm
p116: Basic clustering algorithm
p117: K-means discussion
pp118-: K-means in Spark's MLlib
pp126-129: Finding good value for k
pp130-132: Visualization in R
pp133-136: Normalization
p137: Translating categorical attributed into numerical
pp138-140: Measuring entropy
p141: Final anomaly detection (+ anomaly definition)

In information theory, entropy is a measure of uncertainty or randomness. In the context of clustering, entropy measures the uncertainty or randomness of the labels within each cluster. If all data points in a cluster have the same label, the entropy is low, indicating high purity or homogeneity. Conversely, if the labels within a cluster are mixed or diverse, the entropy is high, indicating low purity or homogeneity.

Chapter 6: Medical Network Analysis in GraphX (pp143-180)

p144: GraphX
pp147-150: Parsing XML data
p151: Absolute frequencies
p152: Co-Occurrences of topics + maximum amount of unique edges in undirected graphs
pp153-154: Hashing for vertices
p155: Vertices
p156: Edges
p157: Network analysis algorithms: - connected components - degree distribution - cliques/triangles & clustering coefficients - diameter & average path length

pp158-161: Connected components
pp162-163: Degree distribution
pp164-169: Digression: filtering out noisy edges (Chi-squared test)
pp170-171: Cliques, triangle counts & clustering coefficients
pp172-177: Average path length
pp178-179: PageRank algorithm
p180: Summary

Chapter 7: Geospatial, Temporal & Streaming Data Analysis (pp181-210)

p182: Temporal & spatial data
p183: Spheres & polygons
p184: GeoJSON
p186: Date & time APIs
pp187-189: Geometric APIs
pp190-191: GeoJSON API
p192: Custom TaxiTrip data structure
p193: *parse* function
pp194-198: Safe parsing & filtering: *Either[left, right]*
pp199-200: Loading boroughs geometry
p201: Combining datasets
p203: "Sessionization" of taxi trips
pp204-205: Secondary sorting
p207: *Sliding* operator
p210: Summary

Chapter 8: Financial Risk Analysis via Monte Carlo Simulations (pp212-242)

p213: Value-at-Risk estimation
p215: Stock markets & returns of investments
p216: Linear regression model
p217: Value-at-Risk & financial risk
p218: Portfolio, stocks, factors
p219: Time series data
p220: Monte Carlo simulations for market conditons & VaR
p221: Multivaraite normal
p222: Datasets for stocks and factors
pp223-224: Parsing time series files
p225: Trimming time series
p226: Auto-completing time series
p227: Sliding window & bi-weekly returns
pp228-229: Plotting factor returns
p230: Fitting the paramters: computing the factor means and co-variances
p231: "Featurizing" factor returns
p232: Training linear-regression model
p233: Parallel sampling
p234: Spark *Dataset* API
p235: Running the sampler
p236: Predicting the average stock return
p237: Calculating the VaR
p240: Conditional VaR
p241: Summary

Strongly typed data structures are those where types are explicitly declared and enforced. Type checking: In strongly typed languages, the compiler performs rigorous type checking at compile time. This prevents operations that are not type-safe, such as trying to add a string to an integer.

Weakly typed data structures allow more flexibility and implicit type conversion. Variables can change types, and operations can be performed on mismatched types through implicit coercion. Implicit Conversions: Weakly typed languages often perform implicit type conversions, which can lead to unexpected behavior and bugs that are harder to trace.