ML-4

1. **Ensemble techniques** combine multiple models to improve predictions (e.g., Random Forest, Boosting).
2. **Bagging** trains models in parallel on bootstrapped samples, then aggregates results (e.g., Random Forest).
3. **Bootstrapping** creates diverse training subsets by sampling with replacement.
4. **Random Forest** uses bagging with decision trees + random feature selection.
5. **Randomization** reduces overfitting by limiting tree correlation (via feature bagging).
6. **Feature bagging** randomly selects subsets of features per tree split.
7. **Decision trees in Gradient Boosting** are weak learners trained sequentially on residuals.
8. **Bagging vs Boosting**:
    - *Bagging*: Parallel, reduces variance.
    - *Boosting*: Sequential, reduces bias.
9. **AdaBoost** adjusts weights for misclassified points in each iteration.
10. **Weak learners** perform slightly better than random guessing (e.g., shallow trees).
11. **Adaptive Boosting** reweights misclassified samples to focus on hard cases.
12. **AdaBoost weight adjustment**: Increases weights for misclassified points.
13. **XGBoost advantages**: Regularization, parallel processing, handling missing data.
14. **XGBoost regularization**: Penalizes complex trees (L1/L2).
15. **Ensemble types**: Bagging, Boosting, Stacking.
16. **Bagging vs Boosting comparison**:

- *Bagging*: Good for high-variance models.
- *Boosting*: Good for high-bias models.

17. **Ensemble diversity**: Using varied models/data reduces collective errors.
18. **Improved performance**: Averages errors, reduces overfitting.
19. **Ensemble bias/variance**:

- *Bias*: Averaging reduces bias (Boosting).
- *Variance*: Aggregation reduces variance (Bagging).

20. **Trade-off**: More models reduce variance but increase computation.
21. **Applications**: Fraud detection (Random Forest), ranking (XGBoost).

22. **Interpretability**: Less interpretable than single models (trade-off for accuracy).
23. **Stacking**: Combines models via meta-learner (e.g., logistic regression).
24. **Meta-learners**: Train on base model predictions (final aggregator).
25. **Challenges**: Computationally expensive, risk of overfitting.
26. **Boosting** iteratively corrects errors of prior models (vs bagging's parallelism).
27. **Boosting intuition**: Focuses on hard-to-predict samples.
28. **Sequential training**: Each model learns from previous errors.
29. **Misclassified handling**: Reweights or resamples difficult points.
30. **Weights**: Prioritize high-error samples in next iteration.
31. **AdaBoost vs Boosting**: AdaBoost is a specific boosting algorithm with weight updates.
32. **AdaBoost weight adjustment**: Doubles weights for misclassified points.
33. Weak learners: Simple models (e.g., depth-1 trees) slightly better than random.
34. Gradient Boosting: Fits trees to residuals (errors) of previous models.
35. Gradient descent: Minimizes loss function by iteratively updating model.
36. Learning rate: Controls contribution of each tree (smaller = more robust).
37. Overfitting handling: Early stopping, shrinkage (learning rate < 0.1).
38. XGBoost vs Gradient Boosting: XGBoost adds regularization, parallel processing.
39. Regularized boosting: Penalizes tree complexity (L1/L2 in XGBoost).
40. XGBoost advantages: Faster, handles missing data, built-in cross-validation.
41. Early stopping: Halts training if validation score doesn't improve.
42. Prevents overfitting: Stops before model fits noise.

43. Hyperparameters: Learning rate, tree depth, subsampling ratio.

44. Challenges: Sensitive to noise, computationally intensive.

45. Convergence: Stops when errors stop improving significantly.

46. Performance improvement: Sequentially corrects errors.

47. Data imbalance: Can worsen minority class errors (use class weights).

48. Applications: Click prediction (XGBoost), medical diagnosis (AdaBoost).

49. Ensemble selection: Chooses subset of models to optimize performance.

50. Interpretability: Feature importance scores (less transparent than single trees).

51. **Curse of dimensionality**: High dimensions increase sparsity, hurt KNN performance.
52. **KNN applications**: Recommendation systems, image classification.
53. **Weighted KNN**: Closer neighbors have higher voting power.
54. **Missing values**: Impute or use distance metrics handling missingness (e.g., Gower).
55. **Lazy vs eager learning**: KNN is lazy (no training, predicts at runtime).
56. **Improving KNN**: Feature scaling, dimensionality reduction, optimal *K*.
57. **KNN regression**: Predicts average (or median) of *K* neighbors.
58. **Decision boundary**: Highly irregular (voronoi tessellation).
59. **Choosing *K***: Cross-validation or elbow method (error vs *K* plot).
60. **Small vs large *K***:
- *Small*: Noisy but flexible.
- *Large*: Smooth but may underfit.
61. **Feature scaling**: Critical for distance-based metrics (e.g., Min-Max).
62. **KNN vs SVM/DT**:
- *KNN*: No training, slow prediction.
- *SVM*: Better for high dimensions.
- *DT*: More interpretable.
63. **Distance metric impact**: Euclidean (scale-sensitive), Manhattan (robust to outliers).
64. **Imbalanced data**: Use weighted voting or resampling.
65. **Cross-validation**: Tunes *K* and distance metrics via grid search.
66. **Uniform vs distance-weighted**:
- *Uniform*: Equal neighbor votes.
- *Weighted*: Closer neighbors matter more.
67. **Computational complexity**: O(*n*) per query (slow for large datasets).

68. **Outlier sensitivity**: Manhattan distance is more robust than Euclidean.
69. **Elbow method**: Pick *K* where error rate stabilizes.
70. **Text classification**: Use TF-IDF vectors + cosine similarity.
71. **PCA components**: Retain 95% variance or use scree plot.
72. **Reconstruction error**: Measures info loss when reducing dimensions.
73. **PCA applications**: Image compression, noise reduction.
74. **PCA limitations**: Linear assumptions, loses interpretability.
75. **SVD**: Generalization of PCA (works on non-square matrices).
76. **LSA**: Applies SVD to term-document matrices (topic modeling).
77. **PCA alternatives**: t-SNE, UMAP, ICA.
78. **t-SNE advantages**: Preserves local structure, good for visualization.
79. **t-SNE vs PCA**: t-SNE captures nonlinear patterns; PCA is linear.
80. **t-SNE limitations**: Computationally heavy, no inverse transform.
81. **PCA vs ICA**: PCA finds uncorrelated components; ICA finds independent sources.
82. **Manifold learning**: Unfolds nonlinear structures (e.g., Isomap).
83. **Autoencoders**: Neural networks for nonlinear dimensionality reduction.
84. **Nonlinear challenges**: Harder to interpret, computationally expensive.
85. **Distance metric impact**: Euclidean for PCA, cosine for text.
86. **Visualization**: Scatter plots (2D/3D), heatmaps.
87. **Feature hashing**: Reduces dimensions via hashing trick (for high-cardinality data).
88. **Global vs local methods**:
- *Global*: PCA preserves overall structure.
- *Local*: t-SNE preserves neighborhood relationships.
89. **Sparsity impact**: May require specialized methods (e.g., sparse PCA).
90. **Outlier impact**: Can distort principal components (use robust PCA).