Meeting 11: Linear Regression, Basis, Regularization

Supervised learning setup

- Goal: learn function $f(x)$ that maps features X to target Y.
- Data: training pairs (x_i, y_i), $i = 1, \ldots, n$.
- Prediction: $\hat{y}_i = f(x_i)$ should be close to y_i.
- Three sets:
  - **Train**: fit parameters (slopes, intercepts, weights).
  - **Validation**: tune hyperparameters (degree, λ, etc).
  - **Test**: used once at very end for final evaluation.

Error measurement

- Point error: $e_i = \hat{y}_i - y_i$.
- Absolute error: $\sum_i |e_i|$.
- Squared error: $\sum_i e_i^2 = \sum_i (\hat{y}_i - y_i)^2$.
- Squared error:
  - Always nonnegative.
  - Differentiable almost everywhere, easier for calculus.
  - Large errors get amplified, so model cares more about them.

Least squares linear regression

- Simple line, one feature:
  - Model: $y = mx + b$.
  - Loss: $\text{RSS}(m, b) = \sum_i (y_i - (mx_i + b))^2$.
  - Geometric view: vertical distances from points to line.
- Multivariate, p features:
  - Data matrix $X \in R^{n \times p}$, target $y \in R^n$.
  - Model: $\hat{y} = X\beta$, where $\beta \in R^p$.
  - Often add intercept column of ones.
  - Loss: $\text{RSS}(\beta) = \|y - X\beta\|_2^2$.
- Closed form when $X^T X$ invertible:

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

- Scikit-learn uses numerical solvers, but concept stays: minimize RSS.

Assumptions / issues

- Relationship should be roughly linear after any feature transforms.
- Features with high linear correlation give nearly singular $X^T X$.
- This inflates variance of $\hat{\beta}$. Small noise leads to large coefficient swings.
- Extreme outliers can dominate fit because of squared error.

Network traffic example (bytes vs packets)

- Task: predict `bytes` from `packets` in NetML flow matrix.
- Scatter plot shows two clusters:
  - Data packets: large bytes, variable packet counts.
  - ACK packets: very small bytes, lower region.
- Squared error gives big flows more influence:
  - Errors on large-byte points dominate RSS.
  - Regression line fits upper cluster better, lower cluster worse.
- Takeaway: single linear model has trouble with multiple regimes.

Fixing ACK artifact

- Option 1: rescale features or use different loss (less effect here).
- Option 2 (better): treat as two-stage pipeline:
  - Stage 1: classify packet as data vs ACK.
  - Stage 2: apply separate linear regression model for each class.
- Lesson: sometimes pipeline of simpler models beats one global model.

Polynomial basis expansion

- Many relationships are nonlinear in raw x, but linear in transformed features.
- For one feature x, create basis:

$$\phi(x) = [1, x, x^2, x^3, \ldots, x^d]$$

- Fit linear regression on $\phi(x)$:

$$\hat{y} = \beta_0 + \beta_1 x + \cdots + \beta_d x^d$$

- Model is still linear in β. Nonlinearity is in feature mapping.
- Can also include interaction terms, for example $x_1 x_2$.

Overfitting from high degree

- Degree d controls flexibility.
- As d increases:
  - Training error always decreases.
  - Beyond a point, validation and test error increase.
- High degree polynomials can oscillate wildly between points.
- Classic bias variance tradeoff:
  - Low degree: high bias, low variance.
  - High degree: low bias, high variance.

Choosing polynomial degree (hyperparameter tuning)

- Never choose degree by looking at test error.
- Procedure:
  - Split training into train and validation sets, or use K fold CV.
  - For degrees d in grid, fit on train, compute error on validation.
  - Plot training and validation error vs d.
  - Pick degree where validation error is smallest (often near U shaped minimum).
- Same pattern applies to almost all model complexity knobs.

Two sources of complexity in linear models

- Basis design: how many transformed features, degree of polynomials, kernels.
- Effective dimension: how many coefficients significantly nonzero.

Ridge regression (L2 regularization)

- Idea: add penalty on large coefficients.
- Objective:

$$\text{RSS}(\beta) + \lambda \sum_j \beta_j^2 = \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2$$

- λ controls strength of regularization (called `alpha` in sklearn).
- $\lambda = 0$ gives ordinary least squares.
- Large λ shrinks coefficients toward zero.
- Benefits:
  - Reduces variance and sensitivity to noise.
  - Handles multicollinearity better.
  - Often improves test error even if training error worsens.

Lasso (L1) and elastic net

- Lasso objective:

$$\|y - X\beta\|_2^2 + \lambda \sum_j |\beta_j|$$

- L1 penalty often produces exact zeros in β (automatic feature selection).
- Elastic net:

$$\|y - X\beta\|_2^2 + \lambda_1 \sum_j |\beta_j| + \lambda_2 \sum_j \beta_j^2$$

- Mixes sparsity from L1 and stability from L2.

Tuning λ

- Hyperparameter like degree.
- Use validation or cross validation on **training data only.**
- Often search over log scale, for example $\lambda \in \{10^{-4}, \ldots, 10^3\}$.
- Choose λ that minimizes average validation error.

Why not one λ_j per feature

- Could define:

$$\text{RSS} + \sum_j \lambda_j \beta_j^2$$

- But:
  - Hyperparameter space becomes d-dimensional.
  - Search becomes combinatorial and expensive.
  - Hard to reason about, no simple regularization path.
- Using one λ encodes "we care about total number / size of coefficients, not which ones".

Kernel regression example (Google CDN paper)

- They encode configuration parameters with radial basis functions.
- Apply linear regression on expanded feature space.
- Real outcome: simple basis expansion plus linear regression can solve production problems.

Meeting 11: Logistic Regression

Model

- Binary labels $y \in \{0, 1\}$.
- Compute linear score $z = w^T x + b$.
- Map to probability:

$$p(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- Decision rule: predict class 1 if $p \geq 0.5$, else class 0.

Interpretation

- Log odds:

$$\log \frac{p}{1 - p} = w^T x + b$$

- Each coefficient w_j is additive effect of feature x_j on log odds.
- Sign of w_j: positive increases odds of class 1, negative decreases.

Training

- **Loss**: negative log likelihood = binary cross entropy.
- For one point:

$$\ell(w) = -[y \log p + (1 - y) \log(1 - p)]$$

- No closed form solution. Use gradient descent or variants.
- Loss is convex in w. Global optimum exists, unlike deep nets.

When logistic regression works well

- Data roughly linearly separable in feature space, or after simple basis expansion.
- DNS example: classify query vs response based on size, since responses are larger.
- Could also classify data vs ACK packets in NetML.

Meeting 12: Trees and Ensembles

Decision trees

- Partition feature space into axis aligned regions by recursive splits.
- Node: test like $x_j \leq t$. Left and right children form subregions.
- Leaf: stores prediction.

Regression trees

- At leaf, prediction is mean of training y_i in that region.
- Split selection: choose feature and threshold that minimize RSS across children.

Classification trees

- Leaves store class probabilities.
- Need impurity measure at node:
  - Let p_k be fraction of class k in node.
  - Misclassification error: $1 - \max_k p_k$.
  - Gini index: $\sum_k p_k(1 - p_k)$.
  - Entropy: $-\sum_k p_k \log p_k$.
- Splits chosen to reduce impurity the most.

Tree growth, depth, and pruning

- Algorithm:
  - Start with root that has all data.
  - At each step, choose best split among all features and thresholds.
  - Recurse until stopping rule (max depth, min samples, no gain).
- Greedy: never revisits past splits.
- Deep trees can fit noise (each leaf may contain few points).
- Cost complexity pruning:

$$\text{RSS}_{\text{tree}} + \alpha \times \text{number of leaves}$$

  or classification version.
- Choose α by cross validation.

Pros and cons of single trees

- Pros:
  - Very interpretable. Path from root to leaf is human readable rule.
  - Work with numeric and categorical features.

  - No scaling needed.
- Cons:
  - High variance, small change in data can change top split.
  - Axis aligned boundaries may be too rigid.
  - Alone, rarely best accuracy.

Ensembles: idea
- Combine many weak or unstable models to form stronger one.
- Need diversity among models for averaging to help.
- For trees, use bagging, random forests, or boosting.

Bagging (bootstrap aggregating)
- For B times:
  - Sample n points with replacement from training data.
  - Train a tree on this bootstrap sample.
- Prediction:
  - Regression: average of tree predictions.
  - Classification: majority vote.
- Reduces variance because individual tree errors average out.

Random forests
- Bagging plus feature randomness at each split.
- At each node:
  - Sample subset of features (for example $\sqrt{d}$ in classification).
  - Only search this subset for best split.
- This decorrelates trees:
  - Without randomness, strong features picked early by all trees.
  - With randomness, different trees follow different paths.
- Overall effect: lower variance, better generalization.
- Feature importance:
  - Measure average impurity decrease when splitting on feature across forest.
  - Gives ranking of informative features.

Boosting (very brief)
- Builds trees sequentially, not independently.
- Each tree trained on reweighted data that emphasizes previous errors.
- Each tree is shallow (stumps or depth few).
- Hyperparameters:
  - Number of trees.
  - Learning rate (how much each tree contributes).
  - Tree depth.

IoT privacy example (Nest camera)
- Input: time series of upload rates from camera.
- Label: motion vs idle.
- Simple threshold on rate works. Trees and forests improve margins, handle noise.
- Privacy risk: even encrypted traffic leaks activity patterns via rates.

Meeting 13: Deep Learning Fundamentals

Why deep learning for network data
- Traditional approach:
  - Hand craft flow features: counts, average packet size, interarrival times.
  - Requires domain expertise and repeated effort for each task.
- Deep learning:
  - Learns internal representations from raw or lightly processed inputs (for example nPrint).
  - Same architecture can be reused for different tasks with new labels.

Artificial neuron
- Inputs $x \in R^d$, weights w, bias b.
- Compute pre-activation:

$$z = w^T x + b$$

- Apply activation:

$$a = f(z)$$

- Without activation, a stack of layers collapses to one linear map.

Activation functions
- **Sigmoid**: $\sigma(z) = 1/(1 + e^{-z})$.
- **Tanh**: $\tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$.
- **ReLU**: $\mathrm{ReLU}(z) = \max(0, z)$.
- Pros and cons:
  - Sigmoid, tanh saturate for large $|z|$, gradients near zero.
  - ReLU has simple piecewise linear shape, no saturation on positive side.
  - ReLU can suffer from dead neurons for negative region.

Feed forward networks
- Layers indexed by ℓ.
- Layer computation:

$$a^{(\ell)} = f(W^{(\ell)} a^{(\ell-1)} + b^{(\ell)})$$

- Input layer: raw features. Hidden layers: intermediate representations. Output layer: predictions.
- Depth: number of layers with parameters. Width: number of units per layer.

Loss functions
- Regression: MSE:

$$\frac{1}{n} \sum_i (\hat{y}_i - y_i)^2$$

- Binary classification: binary cross entropy.
- Multi class: categorical cross entropy with softmax.

Training by gradient descent
- Forward pass: compute outputs and **loss** for minibatch.
- Backward pass:
  - Use chain rule to compute gradients of loss w.r.t weights and biases.
  - Backpropagate errors from output to early layers.
- Update weights:

$$\theta \leftarrow \theta - \eta \nabla_\theta L$$

  where η is learning rate.
- Mini batch gradient descent:
  - Use subset of examples per update (for example 32 or 64).
  - Balance between noisy SGD and full batch.

Vanishing and exploding gradients
- Vanishing:
  - Gradients shrink as they multiply by derivatives less than 1 across layers.
  - Early layers receive tiny updates, learn slowly.
- Exploding:
  - Gradients blow up and cause unstable training.
- Mitigation:
  - ReLU and variants.
  - Careful initialization.
  - Batch normalization.
  - Residual (skip) connections.
  - Gradient clipping.

Overfitting and regularization in deep nets
- Networks with many parameters can memorize training data.
- Tools:
  - L2 weight decay.
  - Dropout: randomly zero hidden units during training.
  - Early stopping based on validation loss.
  - Data augmentation (for example in vision).

When deep learning helps vs traditional models
- Use deep learning when:
  - Large labeled dataset.
  - Complex patterns in high dimensional inputs (for example nPrint bitmaps).
  - Manual feature engineering is hard.
- Use simpler models when:
  - Data small, need interpretability.
  - Good hand crafted features exist.
  - Limited compute.
  - Random forests or logistic regression often enough.

Key Concepts to Remember
Neuron: Weighted sum + bias + activation function Activation functions: Introduce non-linearity (ReLU most common) **Feed-forward network**: Input → hidden layers → output **Forward propagation**: Data flows through network **Backpropagation**: Gradients flow backward to update weights **Gradient descent:** Iterative weight updates to minimize loss **Epochs**: Full passes through training data Vanishing/exploding gradients: Major training challenges **Representation learning**: Networks learn features automatically **Deep learning advantage**: No manual feature engineering needed

Meeting 14: nPrint and Representation Learning

Motivation
- Each network ML project often reimplements:
  - Feature extraction code.
  - Flow definitions and filtering.
  - Formats for storing features.
- This makes results hard to reproduce and compare across papers.

nPrint representation
- Input: raw packet capture (pcap).
- Output: CSV with one row per packet, many columns representing bits. (Fixed-size bitmap where each bit always means the same thing. Solves the alignment problem)
- Bit positions fixed:
  - Certain ranges reserved for IPv4 header.
  - Others for IPv6, TCP, UDP, payload slices, etc.
- Values:
  - 1: bit set.
  - 0: bit clear.
  - −1: field not present at all in this packet.
- −1 is critical to keep alignment when field absent (for example no TCP header).

Typical usage
- Example commands:

```
# Generate nPrint with first 30 bytes of payload
nprint -P 30 input.pcap > out.csv
# Include IPv4 and TCP headers with payload
nprint -P 30 -4 -t input.pcap > out.csv
```

- Steps:
  - Generate nPrint CSVs.
  - Create label file for classification (for example scan vs benign).
  - Join in pandas.
  - Train classifier (random forest, deep net).
  - Inspect feature importance or learned representations. (which bits/header fields matter))

Output:
CSV file where each row = one packet
Each column = one bit position in standardized representation
Typically hundreds of features (bits) per packet, can load directly into pandas DataFrame

Benefits
- Unified representation across tasks. (Don't need to redesign feature extraction)
- Easier reproducibility: same pcap and flags produce identical CSV.
- Automation: Deep learning can learn which bits matter without manual feature design.

Spurious correlations - nPrint includes all header fields indiscriminately
- Models may latch onto patterns that don't generalize
Example: All training examples from one IP address range Model might learn to recognize IP range instead of application behavior (IP instead of attack behavior
- Requires careful data collection, cross-validation, feature importance analysis

Limitations

- Very high dimensional and dense. Storage and compute heavy.
- Represents each packet independently. No inherent sequence or timing encoding.
- For tasks that depend on ordering (for example TCP handshake), must group packets or use sequence models.

Hands-On Activity: nPrint for Scan Detection Part 1: Generate nPrints Install nPrint tool (compile from source) Process pcap files to generate bitmap representations Takes 5-10 minutes to get building properly **Part 2: Train Classifier** Load nPrint CSV into DataFrame Generate labels (scan vs. benign) Train random forest or other classifier Typically random forest works well **Part 3: Analyze Feature Importance** Examine which bit positions (header fields) were most important Interpret what those bits represent in protocol headers Ask: Does this make sense for the task? **Part 4: Experiment with Different Representations** Try different nPrint flags (-4, -6, -t, etc.) Compare performance with different header combinations Understand which headers contain most informative features

Key Takeaways nPrint enables representation learning for network traffic Automates feature engineering but doesn't eliminate need for domain knowledge Provides reproducibility and standardization benefits Large representation size and potential spurious correlations are concerns Feature importance analysis remains critical Hybrid approaches with AI-assisted engineering may be future direction Not a silver bullet, but valuable tool in network ML toolkit

Meeting 15: Dimensionality Reduction and Autoencoders

Unsupervised learning

- Train without using labels as part of objective. (Labels = ground-truth outputs provided during training. Cat/dog)
- Still can evaluate results with labels afterward.
- Here focus on dimensionality reduction and clustering.

Dimensionality reduction goals

- Goal: Represent high-dimensional data with smaller number of dimensions
- Compress features while keeping structure.
- Help visualization in 2D or 3D.
- Reduce noise and remove redundant dimensions.
- Prepare inputs for clustering or classifiers.

PCA: principal component analysis

Goal: Represent high-dimensional data with smaller number of dimensions (Change of basis - representing data as linear combination of different features. Assumes linear relationships in data . Find most important features/directions for describing data points)

- **Principal Component Analysis (PCA)**
  - **Core Concept:**
    - Change of basis – representing data as a linear combination of different features
    - Assumes linear relationships in data
    - Finds most important features/directions for describing data points
  - **Mathematical Intuition:**
    - Two ways to view it:
      1. **Maximum variance**: Find direction that captures the most variance in the data.
      2. **Minimum projection distance**: Minimize distance when projecting points onto a lower-dimensional space.
    - These are equivalent approaches.
  - **Principal Components:**
    - First PC: Direction of maximum variance (PCA finds the orientation where the data looks the most stretched out, because that direction contains the most information (variance).
    - Second PC: Orthogonal to first, captures remaining variance
    - Size of components (eigenvalues) indicates variance in each direction
  - **Choosing Number of Components:**
    - **Scree plot**: Plot variance explained vs. number of components
    - Look for the "elbow" or "knee" in the curve
    - Consider domain knowledge (e.g., if looking for 2 classes, try 2 components)
    - Explained variance: How much of total variance is captured by N components
  - **Spectral Clustering with PCA:**
    - PCA can be used for clustering
    - Each point has components in PC1, PC2, etc.
    - Points with more of PC1 $\rightarrow$ cluster 1; more of PC2 $\rightarrow$ cluster 2
    - Number of PCs chosen = number of clusters
  - **Extensions:**
    - Kernel PCA: Handles non-linear relationships
    - Apply functions to PCA to express non-linear patterns
- Input: centered data matrix $X \in R^{n \times d}$.
- Covariance:

$$S = \frac{1}{n} X^T X$$

- Eigen decomposition of S:
  - Eigenvectors give principal directions.
  - Eigenvalues give variance along each direction.
- Two equivalent views:
  - Directions of maximum variance.
  - Best low-rank linear approximation in least squares sense.
- Projection onto top k PCs gives reduced representation.
- Scree plot: cumulative variance vs number of components.

PCA for clustering and visualization

- Plot first two PCs to see approximate class separation.
- Use top k PCs as input to clustering algorithms (for example K means).

t-SNE

- Nonlinear dimensionality reduction that preserves local neighbor structure.
- Constructs probability distributions over pairs of points in high and low dimensional spaces.
- Optimizes mapping so neighbors in high dimensional space remain neighbors in 2D or 3D.
- Great for visualizing clusters. Distances between clusters not always meaningful.

- **Autoencoders**
  - **Architecture:**
    - Deep neural network with encoder and decoder
    - Encoder: Reduces dimensionality (drops coefficients)
    - Bottleneck: Compressed representation (reduced dimensionality)
    - Decoder: Attempts to reconstruct original input
  - **Training Process:**
    - Goal: Make output match input as closely as possible
    - If decoder can reconstruct from bottleneck, encoder did a good job
    - Decoder is mainly for training the encoder
  - **Advantages over PCA / t-SNE:**
    - No need to think about feature engineering
    - Can work directly with raw data
    - Similar to deep learning vs. random forests in supervised learning
  - **Disadvantages:**
    - Much more computationally expensive
    - Requires more data and training time
  - **Applications:**
    - Same as PCA / t-SNE (visualization, compression, preprocessing)

Meeting 16: Generative AI and Diffusion Models

Synthetic (data that is generated by a machine, not collected from the real world): packets created by a generative model that imitate real traffic.

Motivation for synthetic network traffic

- Real traces:
  - Reveal topology, internal IPs, routing policies, user behavior.
  - Hard to release publicly for privacy and security reasons.
- Synthetic traffic can:
  - Training machine learning models with insufficient real data
  - Augmenting datasets (e.g., scaling 100 traces to 10,000)
  - Testing hardware/software, protocol interoperability; Privacy-preserving data sharing

Earlier approaches

- Simulators: replay or approximate traffic patterns. (Limited utility in ML: copies don't add variation needed for model robustness)
- GAN-based generators:
  - Generator produces samples, discriminator judges real vs fake. (Generate realistic variations of network traffic)
  - Often operate at flow level (aggregate features like counts, bytes).
  - Limited when you need packet-level detail.
- Problems:
  - ML models trained on synthetic then tested on real often perform poorly.
  - Protocol violations, unrealistic patterns.

Diffusion model basics

- Forward process:
  - Gradually add Gaussian noise over T steps to data.
  - Final distribution approaches isotropic Gaussian.
- Reverse process:
  - Learn model that, at each step, predicts noise component.
  - Use this to denoise from step T back to step 0.
- Sampling:
  - Start from random noise.
  - Apply reverse steps to generate sample.
- Advantages over GANs:
  - Training more stable.
  - Less mode collapse.
  - Key point for exam: Diffusion = denoising process, GAN = one-shot generation.

4. Conditional Generation (Control Over Outputs)

Diffusion models can be guided using:

- **Text prompts** (via CLIP embeddings)
- **ControlNet** (extra structure constraints)

Why this matters for networking

Network traffic has **strict structure**:

- IP header must appear before TCP header.
- TCP handshake must obey ordering.

Diffusion will violate these rules unless constrained.

ControlNet forces diffusion to respect structure (similar to guiding a sketch $\rightarrow$ image).

NetDiffusion style pipeline

Step 1: Convert traffic to images

- Each packet becomes a row in a 2D bitmap.
- Bits represent header fields (similar to nPrint but larger).
- Missing fields become $-\mathbf{1}$ (necessary for alignment).
- Up to 1024 packets per image.

Why: Diffusion models expect image-like inputs.

Step 2: Fine-Tune Stable Diffusion with LoRA

- Use pre-trained Stable Diffusion 1.5.
- LoRA = low-rank adapters enabling quick fine-tuning.
- Pair traffic-images with text prompts like "TCP Amazon traffic".

Benefit: Few training samples needed (few-shot learning).

Step 3: ControlNet for Structure

Diffusion alone generates invalid packets. ControlNet enforces:
- Correct header ordering.
- Packet field boundaries.

Similar to: Using an edge map to constrain a generated image.

Step 4: Post-Processing with Dependency Trees
Even after ControlNet, outputs may violate protocol rules. A dependency tree ensures correctness:
Intra-packet fixes:
- Checksums
- Header field consistency

Inter-packet fixes:
- TCP handshake ordering
- Sequence number progression
- Source/dest IP consistency

Goal: Make minimal edits to produce valid PCAP.

6. Evaluation (Key Results)
Statistical Similarity
- 30 percent improvement vs. NetShare in matching real traffic distributions.
- 70 percent improvement in field-level accuracy.

ML Classification Accuracy
Training models entirely on synthetic → testing on real:
- 60 percent improvement vs. baselines.

Why: Packet-level details carry more distinguishable patterns than flow-level data.
Class imbalance handling
- Rare classes (Facebook, Zoom, "play music" from A3) can be synthetically expanded.

Tool compatibility
- Wireshark parses generated PCAPs.
- TCPreplay runs them with no complaints.

Some fields (TCP flags) still imperfect—research ongoing.

7. Fidelity vs Diversity Trade-off (Important Concept)
Goal: Generated traffic should be:
- **Similar enough** to real traffic (fidelity)
- **Different enough** to add new information (diversity)

Two extremes:
- **Too close** → just copies real packets, useless for ML.
- **Too different** → meaningless noise, breaks protocol patterns.

There is no universally accepted metric to measure this balance.
Key exam-level concepts across meetings 11–17
- Linear vs logistic regression, and how squared loss vs cross entropy works.
- Overfitting, bias variance tradeoff, and hyperparameter tuning with validation.
- Trees, bagging, random forests, and why ensembling reduces variance.
- Pros and cons of deep learning vs traditional models on traffic tasks.
- nPrint's role in representation learning and reproducibility.
- PCA, t-SNE, autoencoders for dimensionality reduction.
- High-level understanding of diffusion based generative traffic models.

Exams

Linear Regression Accuracy
- LR does *not* require a strictly linear true relationship.
- Works fine if relationship is approximately linear or after basis expansion.
- So the statement is **No**.

Deep Learning vs. Random Forest (Network Tasks)
- DL often not better because:
  - Network features already meaningful → RF separates classes well.
  - DL can overfit small or tabular datasets.
  - RF handles nonlinear splits naturally; no scaling needed.
  - DL complexity unnecessary when features already encode semantics.

PCA: Choosing Number of Components (k)
- Use a **scree plot**: variance explained per component.
- Choose k at the **elbow** where added components give diminishing returns.

Dimensionality Reduction (PCA) • Helps visualize high-dim data in 2–3 dims. • Reduces model complexity + noise. • Networking use case: anomaly detection or understanding structure in traffic. (PCA can reveal clusters: PC1/PC2 show directions of largest variance. Points separated along PCs may indicate different groups. Anomalies often appear as outliers on one PC (e.g., unusually large PC2 value).)
K-means vs density clustering: K-means assumes spherical clusters and fails on non-linear shapes. Example: donut-shaped cluster with a small cluster inside. K-means splits the donut incorrectly; DBSCAN finds both clusters because it groups points by density, not distance to a center.
nPrint Advantages:
- No manual feature engineering required; packets converted to aligned bit vectors.
- Standardized representation makes experiments reproducible across datasets.

nPrint Limitations:
- Cannot represent temporal features (interarrival time, ordering, flow dynamics).
- Packet size represented inefficiently due to fixed-width bit padding.

Spurious Correlations in nPrint:
- nPrint encodes all header bits, including non-semantic fields. (true in dataset, but not true in general)
- Models may learn dataset-specific artifacts (e.g., TTL values tied to topology).
- Learned feature may not generalize; model is distinguishing environments, not traffic behavior.

Variable-Length Flow Features
- Flow features like interarrival times or packet-size sequences differ in length because flows have different numbers of packets.
- ML models require fixed-size feature vectors; variable-length sequences cannot be stacked into one matrix.
- NetML normalizes lengths by padding shorter sequences with zeros (and truncating long ones) so all flows have same number of rows.

Overfitting Feature Example
- Spurious features like source IP range, TTL values, or dataset-specific ports can cause overfitting.
- These features correlate with labels in the training environment but do not generalize to new networks.
- Model memorizes artifacts of the dataset → fails on test data.

Random Forest Feature Types RFs can handle mixed feature types (numeric + categorical) because tree splits naturally operate on thresholds or category equality. No special preprocessing needed.
Deep Learning vs Simpler Models
- DL does not always outperform RF/kNN/logistic models.
- DL needs large datasets; many networking datasets are small → overfitting.
- Semantic features (flow duration, sizes, rates) often capture signal well.
- nPrint is extremely high dimensional; simpler models may generalize better.

Unsupervised Learning Applications
- Detect anomalies (unusual traffic volumes/destinations)
- Detect changes that may indicate failures
- Cluster traffic traces into groups (e.g., botnet detection)
- Not used for identifying applications (requires labels)

Evaluating Unsupervised Models
- Training uses no labels, but evaluation can use labels if available.
- After clustering, match clusters to true classes.
- Then compute precision, recall, F1, purity, etc.
- Cannot use precision/recall to guide the unsupervised training process.

When Transformers outperform nPrint + simple models
- nPrint no temporal order; simple classifiers see packets independently.
- Transformers learn long-range sequences and protocol semantics.
- Examples: Detecting DNS lookup followed by TCP connection (multi-step behavior). Multi-stage attacks (scan → exploit → exfiltration). Web/app behavior classification requiring sequence patterns

Linear Regression → models straight-line (first-degree linear) relationships. **Logistic Regression** → used for classification; predicts probabilities for discrete outcomes. **Random Forest advantage** → reduced overfitting due to many averaged trees. **Random Forest randomness** → (1) bootstrap sampling (Each tree trains on a random sample of the data (sampling with replacement), (2) random subset of features at each split. Best visualization technique → t-SNE (preserves local structure, great for 2D/3D plots). **Dimensionality reduction** improves robustness by: • Removing noisy/irrelevant features • Reducing overfitting • Forcing model to learn main structure (e.g., PCA components)
DBSCAN → density-based clustering; finds arbitrary shapes + detects outliers.
K-means → does NOT always give same clusters (depends on random initialization).
Valid dendrogram clusters → must be complete subtrees. Take any merge point (a horizontal bar) in the dendrogram. If you "cut" the tree right at that bar, everything hanging under that cut is a subtree → and that leaf set is a valid cluster.
Generative models for QoE → create **synthetic traffic** to increase data diversity, expands rare cases, fix class imbalance, reduce overfitting, and improve generalization to new network conditions.

Generated
Linear Regression and Regularization: Polynomial expansion lets linear models fit non linear patterns; higher degree polynomials increase overfitting; expansion creates extra feature columns; Ridge uses L2 penalty to shrink weights; L2 reduces reliance on many features.
Decision Trees and Random Forests: Random forests use bootstrap sampling; random feature selection at splits; trees best for interpretability; feature importance helps debugging and feature selection.
Neural Networks and Deep Learning: ReLU avoids saturation and helps gradients; ReLU is efficient; ReLU gives sparse activations; sigmoid and tanh cause vanishing gradients; deep learning does not always outperform simpler models when semantic features already separate classes.
nPrint Representation Learning: In nPrint, 1 means bit present; 0 means bit absent; -1 means header field not present; -1 keeps bit positions consistent; benefits include reproducibility, removal of spurious correlations, deep learning compatibility, standardized representation.
Dimensionality Reduction: t-SNE captures non linear structure; autoencoders learn features directly from raw data and capture non linear patterns.
Diffusion Models and Synthetic Traffic: Use cases include limited labels, privacy preserving sharing, class imbalance; fidelity measures similarity; diversity measures variation; balance between both is needed.
Supervised Learning Basics: Linear regression fits first degree linear relationships; logistic regression predicts probabilities; random forests reduce overfitting; random forests add randomness via bootstrap sampling and random feature subsets.
Unsupervised Learning Basics: t-SNE is best for visualization; dimensionality reduction improves robustness by removing noise and reducing complexity.
Unsupervised Learning Applications: Detect unusual volumes; detect changes signaling failures; cluster traces into behavior groups.
Evaluation of Unsupervised Models: Precision and recall can be used when labels exist for evaluation.
Density Based Clustering: Better than k means for non spherical clusters; handles varying density; detects outliers and nested clusters.
Transformers for Network Tasks: Useful when relationships span multiple flows; capture long range temporal and semantic context.