IMEN574/IDSC574 Programming for Data Science

14. **Review Session**

Spring 2026

Instructor: **Sangwoo Mo**

Today's Roadmap

A AI & Prompting
L2–L4

B AI-Assisted Dev
L5–L6

C Python & Data
L7–L8

D Regression
L9

E Classification
L10

F Clustering
L11

G Dim. Reduction
L12

H Interpretable ML
L13

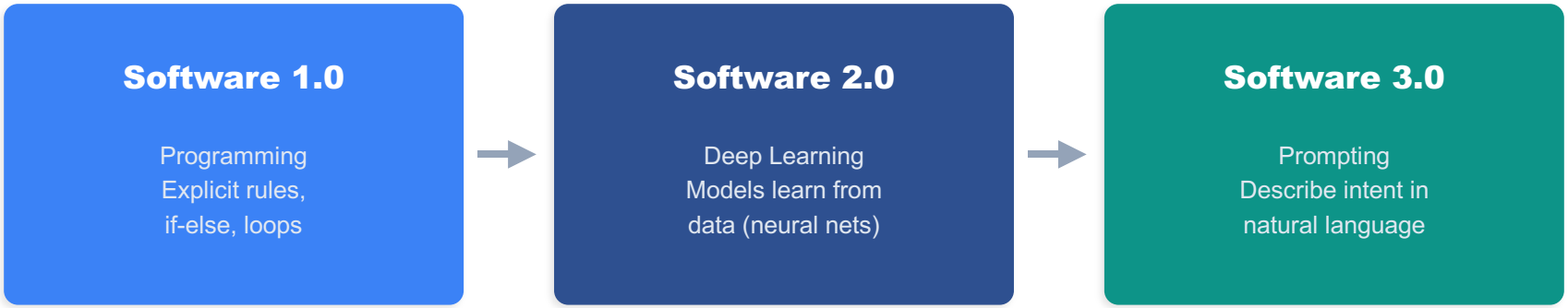
A

AI & Prompting

Lectures 2–4: Chatting with AI, Coding with AI, LLM Prompting

The Evolution of Software

How we build software has fundamentally changed — three paradigms define this shift.



Karpathy (2017) coined 'Software 2.0' — neural networks as a new way to write programs. LLMs take it further: the program is just a prompt.

How Modern LLMs Work

From architecture to training — the key ingredients behind GPT, Claude, and Gemini.

Transformer Architecture

Self-attention lets each token attend to every other — capturing long-range dependencies. Replaced RNNs/LSTMs. (Vaswani et al., 2017)

Scaling Laws

Performance improves predictably with more parameters, data, and compute (Kaplan et al., 2020). Drives the push for ever-larger models.

Training Pipeline



LLMs are next-token predictors — they generate text one token at a time, conditioned on everything before it.

Prompting Techniques

The way you frame a prompt dramatically affects output quality.

Zero-Shot

Give the task directly.
No examples needed.

```
"Translate this to
 French: ..."
```

Best for clear, simple tasks

Few-Shot

Provide input-output
examples before the query.

```
"cat -> gato
 dog -> perro
 bird -> ?"
```

Learns pattern from context

Chain-of-Thought (CoT)

Elicit step-by-step
reasoning before answer.

```
"Let's think step
 by step..."
```

Math, logic, multi-step

Other Techniques

System prompts (set persona/rules), role prompting, structured output (JSON/XML), prompt chaining. The meta-lesson: prompt engineering is about communicating clearly with the model.

Vibe Coding: Power and Limits

Karpathy (2025): writing software by describing what you want in natural language.

What AI Can Do

- Generate boilerplate and scaffolding fast
- Prototype ideas from natural language
- Translate between languages/frameworks
- Write tests, docs, repetitive code

Why Fundamentals Still Matter

- AI makes mistakes — you must debug
- Architecture needs human judgment
- Security, edge cases, performance
- Can't review what you don't understand

AI amplifies your abilities — but only if you have the fundamentals to direct and verify it.

B

AI-Assisted Development

Lectures 5–6: Modern IDE, Coding Agents

Two Paradigms for AI-Assisted Coding

The IDE assists your workflow; the agent takes over the workflow.

The Modern IDE

`Editor | Terminal | Debugger | AI`

Bundles editing, debugging, terminal, and AI completion into one workspace. You write code; the IDE accelerates with suggestions, navigation, and refactoring.

You drive, AI assists

Coding Agents



Repeat until done

A coding agent (Claude Code, Devin) autonomously plans, writes, debugs, and tests code. You set the goal and review the output.

AI drives, you review

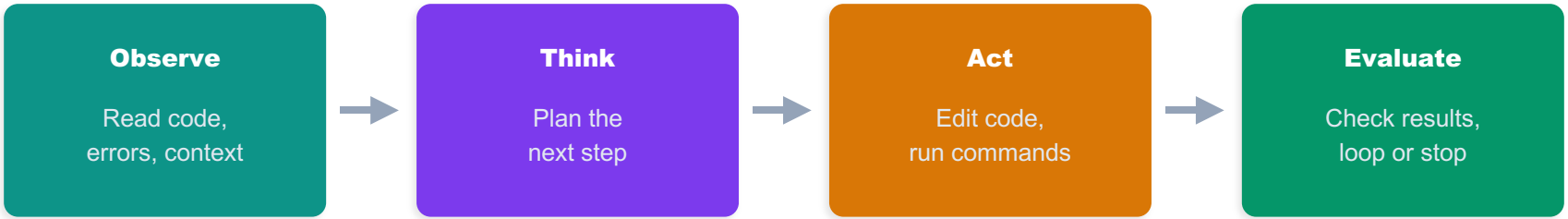
MCP: Connecting Agents to the World

For agents to do real work, they need access to external tools.



MCP provides one protocol connecting any agent to any tool. Instead of custom integrations, agents discover and call tools through a uniform interface.

The Agentic Loop



C

Python & Data Tools

Lectures 7–8: Python OOP, NumPy & Pandas

Object-Oriented Programming

Every major DS library (scikit-learn, PyTorch, Pandas) is built with OOP.




```
model.fit(X_train, y_train)      # Same interface for ALL models
model.predict(X_test)            # Polymorphism in action
```

Abstract Base Class (ABC)

Defines interface. Forces subclasses to implement .fit().

Inheritance

Concrete classes inherit from ABC. Reuse interface, provide implementation.

Encapsulation

Bundle data + behavior. Hide internal state, expose clean methods.

Polymorphism

Same interface, different classes. Duck typing: if it .fit()s, it works.

Advanced Python Patterns

Patterns that make your DS code professional and maintainable.

MixIn

Inject reusable functionality via multiple inheritance.

```
class LogMixin:
  def log(self, msg):
    print(f'[LOG] {msg}')
```

Keeps class hierarchies clean and composable

Decorator

Wrap a function with extra behavior without modifying it.

```
@timer
def train(model):
  model.fit(X, y)
```

Used for timing, caching, logging, authentication

Generator

Lazy evaluation with yield. Essential for large datasets.

```
def batches(data, n):
  for i in range(0, len(data), n):
    yield data[i:i+n]
```

PyTorch DataLoader uses generators internally

Context Manager & Type Hints

with ensures cleanup. Type hints = self-documenting code.

```
with open('data.csv') as f:
  data = f.read()
def process(x: int) -> str:
```

Standard in production Python code

NumPy: Vectorized Computing

NumPy replaces slow Python loops with fast, compiled array operations.

Why Vectorize?

Slow: Python Loop

```
for i in range(len(a)):
  c[i] = a[i] + b[i]
```

Fast: NumPy (100-150x faster)

```
c = a + b  # Compiled C + SIMD
# No loops needed!
```

Core Operations

Broadcasting

Auto-expand dims for element-wise ops

Indexing

Boolean masks, fancy indexing

Linear Algebra

np.dot, np.linalg, SVD, eigendecomp

Random

np.random for reproducible sampling

If you're writing a for-loop over a NumPy array, you're almost certainly doing it wrong.

Pandas & Tabular Data

The standard tool for loading, exploring, cleaning, and transforming structured data.

DataFrame

	name	age	score
0	Alice	25	92.5
1	Bob	30	88.1
2	Carol	28	95.3

Rows = observations, Columns = features

Built on NumPy. Supports loc/iloc, groupby, merge, pivot, method chaining.

Data Wrangling

Missing values: .isna(), .fillna(), .dropna()

GroupBy: split-apply-combine pattern

Merge/Join: combine on shared keys

Apply: custom functions row/col-wise

Best Algorithm for Tabular Data?

Gradient-boosted trees (XGBoost, LightGBM) dominate. They handle mixed types, missing values, nonlinear relationships naturally. Neural nets struggle on tabular data — tree-based methods win Kaggle tabular competitions.

D

Regression

Lecture 9: Linear Regression, Gradient Descent, Regularization

Linear Regression

Predict a continuous value. Start simple, add complexity if needed.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \qquad \text{Minimize MSE} = (1/n)\,\Sigma(y_i - \hat{y}_i)^2$$

Closed-Form: Normal Equation

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

Exact solution in one step. No hyperparameters.

But $O(p^3)$ — impractical for many features.

Iterative: Gradient Descent

$$\beta \leftarrow \beta - \eta \cdot \nabla L$$

Update weights iteratively. SGD uses mini-batches, scales to any size. Standard for all modern ML/DL.

SGD Learning Rate

Too Small

Slow convergence

→

Just Right

Converges well

→

Too Large

Diverges!

Bias-Variance Tradeoff

The fundamental tension in all of machine learning.

Simple Model — **Complex Model**

High Bias (Underfit)

Too simple to capture patterns.

Low variance, but systematic errors.

Ex: linear model on nonlinear data.

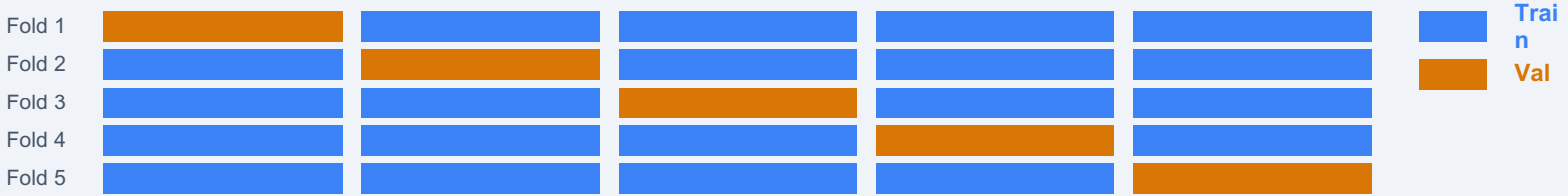
High Variance (Overfit)

Memorizes noise in training data.

Small data changes = different model.

Ex: deep tree on small dataset.

How to Detect: Cross-Validation



Split data into K folds. Each takes a turn as validation set. Average scores to estimate generalization error.

Regularization & Feature Management

Too many features? Regularize or select.

Feature Selection

Keep a subset of original features — interpretability preserved. Forward/backward selection, or Lasso (L1).

Feature Extraction

Create new features as combinations of originals (e.g., PCA). Compresses info, but original meanings lost.

Lasso vs Ridge vs ElasticNet

	Penalty	Effect	Best When
Lasso (L1)	$\Sigma \lvert \beta j \rvert$	Coefficients become exactly 0 = feature selection	Many irrelevant features
Ridge (L2)	$\Sigma \beta j^2$	Coefficients shrink but never reach 0	Multicollinearity, all features matter
ElasticNet	L1 + L2	Combines both: sparsity + stability	Unsure which to use (practical default)

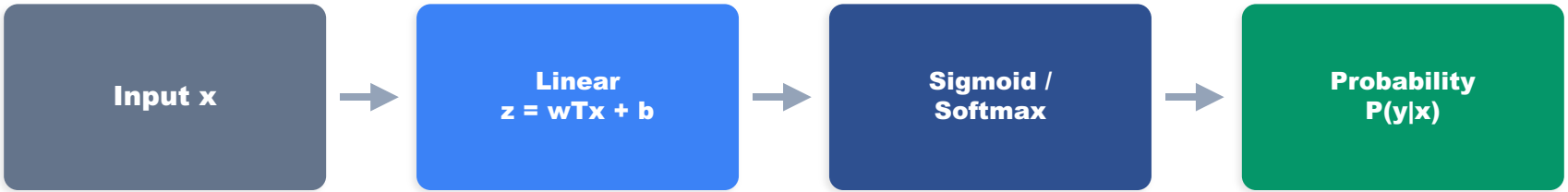
E

Classification

Lecture 10: Logistic Regression, SVM, Ensemble Methods

Classification: From Linear to Probability

Predict discrete categories instead of continuous values.



Binary: Sigmoid

σ(z) = 1/(1+e^-z) maps to (0,1). Output > 0.5 = positive. Threshold is tunable.

Multi-class: Softmax

Converts K logits into probability distribution. Each class in (0,1), all sum to 1.

Cross-Entropy Loss

L = -Σ yi log(yi_hat). Penalizes confident wrong predictions. Convex = GD finds global min.

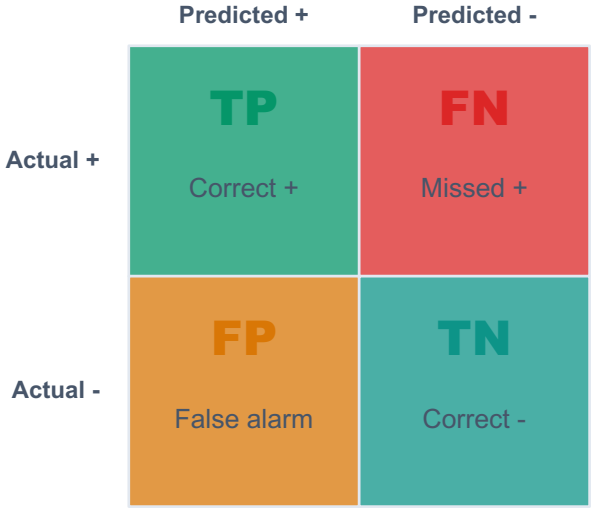
SVM: Max-Margin

Find boundary with maximum margin. Kernel trick for nonlinear boundaries. Good for small datasets.

Evaluation: Beyond Accuracy

Accuracy alone can be misleading (e.g., 99% on 99% negative data).

Confusion Matrix



Key Metrics

Precision `TP / (TP + FP)`

Of predicted positives, how many correct?

Recall `TP / (TP + FN)`

Of actual positives, how many caught?

F1-Score `2PR / (P + R)`

Harmonic mean of precision and recall

ROC-AUC `Area under ROC`

Ranking quality across all thresholds

Choose the metric that matches your cost: missed fraud (recall) vs false alarms (precision).

Ensemble Methods

Combine many weak learners into a strong one.

Decision Tree

Easy to interpret
but overfits

Bagging (Variance Reduction)

Train trees on bootstrapped data with random feature subsets. Aggregate by voting.

Random Forest: decorrelated trees, much lower variance. Robust default.

Boosting (Bias Reduction)

Train trees sequentially — each corrects previous errors.

XGBoost/LightGBM: regularization, handles missing values. Go-to for tabular data.

When to Use What?

	Random Forest	XGBoost	Deep Learning
Tuning	Minimal	Careful	Extensive
Overfit Risk	Low	Medium	High
Best For	Quick, robust default	Max tabular perf	Images, text, audio

F

Clustering

Lecture 11: K-Means, GMM, DBSCAN, Evaluation

Clustering: Finding Structure Without Labels

Unsupervised learning — no labels, just data.

! Feature scaling required for distance-based methods (K-Means, KNN) — otherwise large-range features dominate.

K-Means



Assign to nearest centroid, update, repeat (EM-style).

Spherical clusters only.
Must specify K.
Elbow / silhouette to choose K.

GMM



Soft probabilistic assignments. Each cluster = Gaussian with own mean + covariance.

Elliptical clusters. Also EM.
BIC/AIC to choose K.

DBSCAN



Density-based: dense regions separated by sparse areas.

No K needed. Finds outliers.
Arbitrary shapes (moons, rings).
Params: eps, min_samples.

Evaluation: Silhouette score (how well-separated?), internal vs external metrics. No ground truth = domain knowledge matters.

G

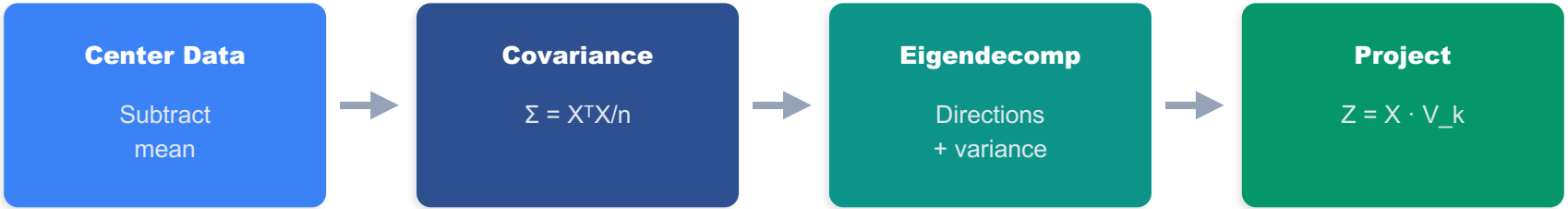
Dimensionality Reduction

Lecture 12: PCA, t-SNE, UMAP, Feature Selection

PCA: Principal Component Analysis

Find directions that maximize variance — compress data while preserving information.

PCA Pipeline



Key Properties

Linear

Finds linear projections only. Fast, interpretable (loadings). Good for preprocessing.

Variance-Based

Explained variance ratio tells how much info each PC captures. Scree plot to choose K.

Solved via SVD

Numerically more stable than eigendecomp. Standard implementation in practice.

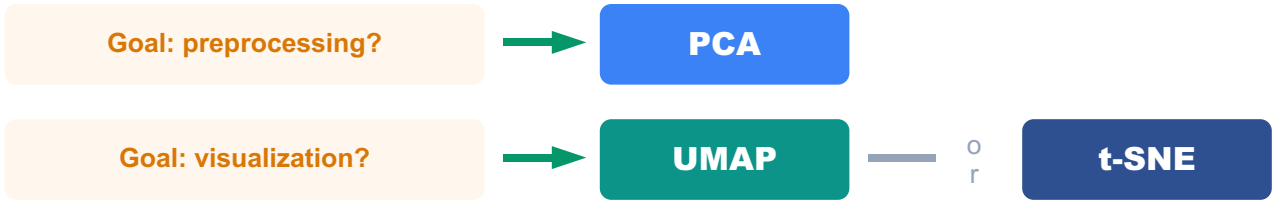
PCA is a feature extraction method — new features are combinations of originals, losing original meanings.

Nonlinear DR: t-SNE & UMAP

When linear methods aren't enough — preserving local structure for visualization.

	PCA	t-SNE	UMAP
Type	Linear	Nonlinear	Nonlinear
Preserves	Global variance	Local neighborhoods	Local + some global
Speed	Fast	Slow ($O(n^2)$)	Fast
Deterministic?	Yes	No	Mostly
Use For	Preprocessing, compression	Visualization only	Visualization + general DR
Interpretable?	Yes (loadings)	No	No

Which method should I use?



H

Interpretable ML

Lecture 13: Why Interpretability, Feature Attribution, Example-Based

Why Interpretability Matters

Accuracy alone isn't enough — we need to understand why.

T

Trust

Stakeholders won't deploy what they can't understand.

D

Debugging

Is the model right for the right reasons? Catch spurious correlations.

R

Regulation

GDPR requires explanations for automated decisions.

I

Discovery

Which features actually drive the outcome? ML as a lens.

Accuracy–Interpretability Spectrum



More Interpretable — More Accurate

Complex model isn't always better — depends on data size, problem complexity, and transparency needs. Start simple.

Feature Attribution Methods

How much did each feature contribute to this prediction?

Feature Importance Ranking



Permutation Importance

Shuffle a feature, see how much accuracy drops. Global method.

PDP / ICE

Plot how output changes as one feature varies. Marginal effects.

SHAP

Shapley values from game theory. Fair per-feature scores. Gold standard.

LIME

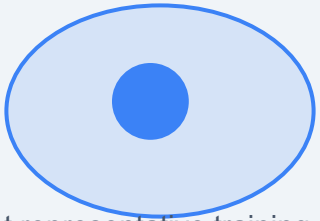
Local linear model around one prediction. Fast, intuitive.

Use multiple methods together — each reveals different aspects of model behavior.

Example-Based Explanations

What does this prediction look like in terms of known examples?

Prototype



Most representative training instance per class.

"This patient looks like a typical case of X."

Counterfactual



Minimal change that flips the prediction.

"If income were $5K higher, loan approved."

Influence Functions



Which training points most influenced this prediction?

Useful for debugging bad predictions.

Feature attribution for stakeholder reports, examples for intuition, counterfactuals for actionable advice.