Model Selection & Ensembles

Choosing, Tuning, and Combining Models

Nipun Batra | IIT Gandhinagar

The Story So Far

We know several algorithms:

- Linear/Logistic Regression
- Decision Trees
- K-Nearest Neighbors

But how do we choose? And can we do better?

Today's Agenda

1. **Bias-Variance Tradeoff** - Why models fail
2. **Cross-Validation** - Better evaluation
3. **Hyperparameter Tuning** - Finding the best settings
4. **Ensemble Methods** - Combining models
5. **Unsupervised Learning** - When there are no labels

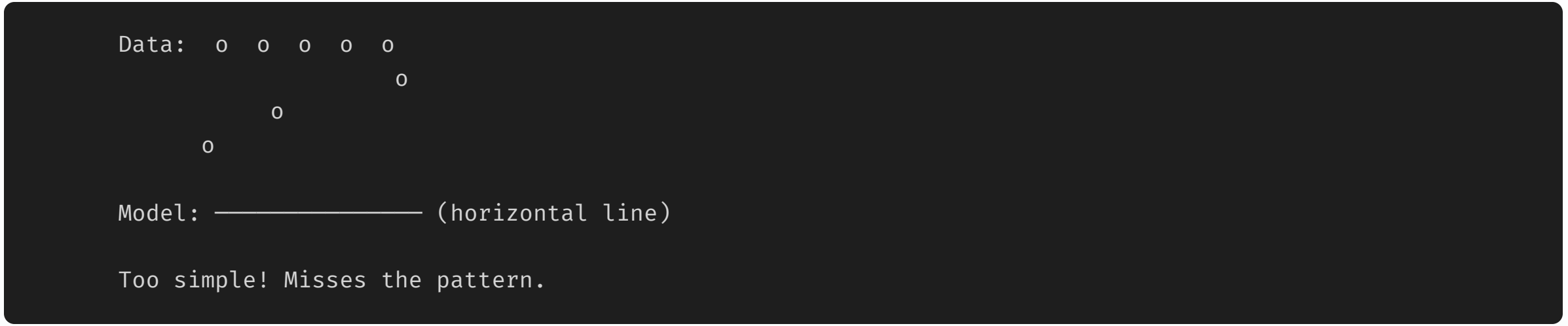
Part 1: Bias-Variance Tradeoff

Why Models Fail

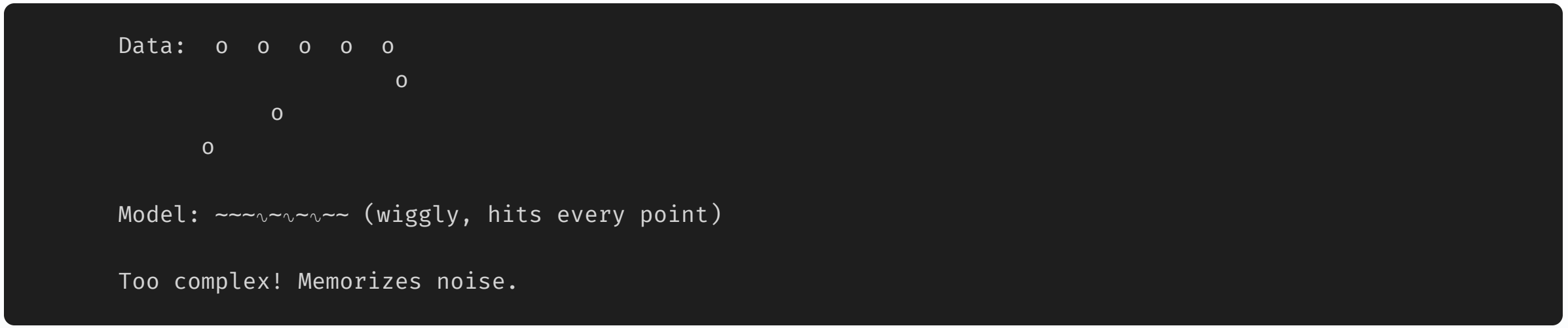
Two Ways to Be Wrong

Type	The model...	Example
High Bias	Is too simple	A horizontal line for curved data
High Variance	Is too complex	Wiggly line that hits every point

Underfitting (High Bias)



Overfitting (High Variance)



The Goldilocks Zone

Bias vs Variance Mathematically

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Term	Meaning
Bias²	Error from wrong assumptions
Variance	Sensitivity to training data
Noise	Random error we can't reduce

The Tradeoff

Model Complexity	Bias	Variance
Very simple	High	Low
Just right	Medium	Medium
Very complex	Low	High

> We can't minimize both simultaneously — we must find the balance!

How to Detect?

Symptom	Diagnosis	Solution
High train error, high test error	Underfitting	More complex model
Low train error, high test error	Overfitting	Simpler model, more data
Low train error, low test error	Good fit!	Ship it!

Learning Curves

Plot error vs training set size:

If gap between curves: OVERFITTING
If both high: UNDERFITTING

Part 2: Cross-Validation

Better Evaluation Than Train/Test Split

Problem with Single Train/Test Split

Issue: Results depend on which data ends up in test set.

Random Seed	Test Accuracy
seed=1	92%
seed=2	87%
seed=3	95%

Which one should we believe?

K-Fold Cross-Validation

Idea: Use every data point for both training AND testing!

```
Fold 1: [Test] [Train] [Train] [Train] [Train]
Fold 2: [Train] [Test] [Train] [Train] [Train]
Fold 3: [Train] [Train] [Test] [Train] [Train]
Fold 4: [Train] [Train] [Train] [Test] [Train]
Fold 5: [Train] [Train] [Train] [Train] [Test]

Final score = Average of all 5 test scores
```

K-Fold in sklearn

```python
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(max_depth=5)

# 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5)

print(f"Scores: {scores}")  # [0.92, 0.88, 0.91, 0.90, 0.89]
print(f"Mean: {scores.mean():.2f}")  # 0.90
print(f"Std: {scores.std():.2f}")   # 0.01
```

Why Cross-Validation?

Benefit	Explanation
More reliable	Average over multiple splits
Uses all data	Every point is tested once
Measures variance	Std shows model stability
Standard practice	Everyone uses it!

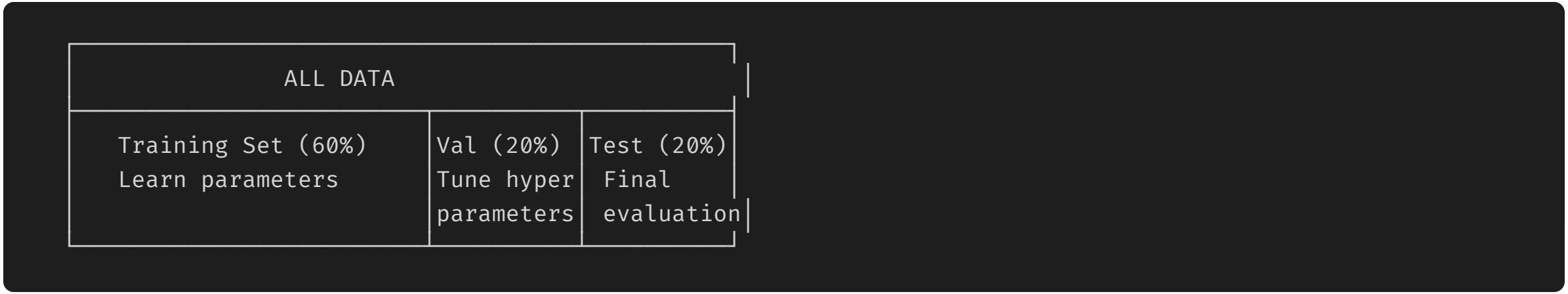
Common K Values

K	Name	When to Use
5	5-Fold	Most common default
10	10-Fold	More reliable estimate
n	Leave-One-Out	Very small datasets

Rule: Larger K = more reliable but slower.

Train/Validation/Test Split

For model selection, we need **three** sets:



Part 3: Hyperparameter Tuning

Finding the Best Settings

Parameters vs Hyperparameters

Type	Learned by	Example
Parameters	The algorithm	Weights in linear regression
Hyperparameters	You (before training)	max_depth in decision tree

Common Hyperparameters

Model	Hyperparameter	Controls
Decision Tree	max_depth	Tree complexity
K-NN	n_neighbors (K)	How many neighbors
Logistic Reg	C	Regularization strength
Random Forest	n_estimators	Number of trees

Grid Search

Try **all combinations** of hyperparameter values:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# Define search space
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_leaf': [1, 5, 10]
}

# Search!
grid = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5)
grid.fit(X_train, y_train)

print(f"Best params: {grid.best_params_}")
print(f"Best score: {grid.best_score_:.2f}")
```

Grid Search Results

```
# All results
results = pd.DataFrame(grid.cv_results_)
print(results[['param_max_depth', 'param_min_samples_leaf',
               'mean_test_score']])

# Use best model
best_model = grid.best_estimator_
predictions = best_model.predict(X_test)
```

Randomized Search

When search space is large, try **random combinations**:

```
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    'max_depth': range(1, 20),
    'min_samples_leaf': range(1, 50)
}

# Try 20 random combinations
random_search = RandomizedSearchCV(
    DecisionTreeClassifier(), param_dist, n_iter=20, cv=5
)
random_search.fit(X_train, y_train)
```

Part 4: Ensemble Methods

The Wisdom of Crowds

The Key Insight

One model can be wrong. Many models are often right!

Single Expert	Panel of Experts
High risk of error	Errors cancel out
Biased perspective	Diverse viewpoints
One decision	Collective wisdom

Two Main Strategies

Strategy	Idea	Method
Bagging	Average many similar models	Random Forest
Boosting	Build models that fix previous errors	XGBoost

Bagging: Bootstrap Aggregating

Steps:

1. Create multiple **random subsets** of training data (with replacement)
2. Train a model on each subset
3. **Average** their predictions (regression) or **vote** (classification)

Random Forest

Decision Trees + Bagging + Feature Randomization

```
from sklearn.ensemble import RandomForestClassifier

# 100 trees, each trained on random data subset
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)

accuracy = model.score(X_test, y_test)
```

Why Random Forest Works

Feature	Benefit
Many trees	Reduce variance
Random samples	Different perspectives
Random features	Decorrelate trees
Voting	Robust to outliers

> Random Forest is often the best "out of the box" model!

Feature Importance

Random Forest tells you which features matter:

```
import pandas as pd

importance = pd.DataFrame({
    'feature': feature_names,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)

print(importance.head())
```

Boosting: Learning from Mistakes

Steps:

1. Train a model
2. Find where it makes errors
3. Train next model to **focus on errors**
4. Repeat, combine all models

Each model **boosts** the performance!

Gradient Boosting in sklearn

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3
)
model.fit(X_train, y_train)
```

XGBoost: The Competition Winner

```
# Install: pip install xgboost
from xgboost import XGBClassifier

model = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5
)
model.fit(X_train, y_train)

# Often wins Kaggle competitions!
```

Bagging vs Boosting

Aspect	Bagging	Boosting
Reduces	Variance	Bias
Training	Parallel (fast)	Sequential (slower)
Risk	Less overfitting	Can overfit
Example	Random Forest	XGBoost, LightGBM

When to Use What?

Scenario	Best Choice
Quick baseline	Random Forest
Competition winning	XGBoost / LightGBM
Interpretability needed	Single Decision Tree
Small data	Simpler models
Lots of data	Deep Learning

Part 5: Unsupervised Learning

When There Are No Labels

No Labels? No Problem!

Sometimes you don't have labels:

- Customer segmentation
- Anomaly detection
- Data exploration

Unsupervised learning finds structure without labels.

Clustering: Find Groups

Goal: Group similar data points together.

Algorithm	How It Works
K-Means	Assign to nearest centroid
DBSCAN	Density-based regions
Hierarchical	Build tree of clusters

K-Means Clustering

```
from sklearn.cluster import KMeans

# Find 3 clusters
model = KMeans(n_clusters=3, random_state=42)
model.fit(X)

# Which cluster is each point in?
labels = model.labels_

# Where are the cluster centers?
centers = model.cluster_centers_
```

K-Means Algorithm

1. **Initialize:** Place K random centroids
2. **Assign:** Each point to nearest centroid
3. **Update:** Move centroids to cluster means
4. **Repeat** until convergence

Choosing K

Elbow Method: Plot inertia vs K

```
inertias = []
for k in range(1, 11):
    model = KMeans(n_clusters=k)
    model.fit(X)
    inertias.append(model.inertia_)

# Plot and look for "elbow"
plt.plot(range(1, 11), inertias)
plt.xlabel('K')
plt.ylabel('Inertia')
```

Dimensionality Reduction

Problem: Can't visualize 100 features!

Solution: Reduce to 2-3 dimensions.

Method	What It Does
PCA	Linear projection, maximize variance
t-SNE	Non-linear, preserve local structure
UMAP	Non-linear, fast, preserves global structure

PCA Example

```
from sklearn.decomposition import PCA

# Reduce to 2 dimensions
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)

# Now we can plot!
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=labels)
plt.xlabel('PC1')
plt.ylabel('PC2')
```

t-SNE for Visualization

```
from sklearn.manifold import TSNE

# Better for visualization
tsne = TSNE(n_components=2, random_state=42)
X_2d = tsne.fit_transform(X)

plt.scatter(X_2d[:, 0], X_2d[:, 1], c=labels)
```

> t-SNE is great for visualizing high-dimensional data like MNIST digits!

Anomaly Detection

Goal: Find the "odd ones out"

```
from sklearn.ensemble import IsolationForest

# Fit on normal data
model = IsolationForest(contamination=0.1)
model.fit(X)

# Predict: 1 = normal, -1 = anomaly
predictions = model.predict(X_new)
```

Summary: Model Selection & Ensembles

1. **Bias-Variance Tradeoff** - Balance simplicity and complexity
2. **Cross-Validation** - More reliable than single split
3. **Hyperparameter Tuning** - Use GridSearchCV or RandomizedSearchCV
4. **Ensembles** - Combine models for better performance
    - Bagging → Random Forest
    - Boosting → XGBoost
5. **Unsupervised** - Clustering, dimensionality reduction

Key Takeaways

> **"No Free Lunch"** - There's no universally best model.

Step	What to Do
1	Start with simple baseline
2	Use cross-validation
3	Tune hyperparameters
4	Try ensembles if needed
5	Check for overfitting

You Can Now Select & Tune Models!

Next: Neural Networks - Deep Learning Begins

Lab: Cross-validation, hyperparameter tuning, Random Forest

"Ensemble methods: Because two heads are better than one!"

Questions?