■ **Course:**	CS109A: Introduction to Data Science
■ **Lecture:**	Lecture 25: Blending, Stacking, and Mixture of Experts
■ **Instructors:**	Pavlos Protopapas, Kevin Rader, Chris Gumb
■ **Objective:**	Learn advanced ensemble methods that combine heterogeneous models, understand meta-learning, and explore the Mixture of Experts architecture used in modern LLMs

Contents

1 The Big Picture: Combining Different Models

Key Summary

This final lecture introduces **advanced ensemble methods** that go beyond Bagging and Boosting:

- **Previous methods** (Bagging, Random Forest, Boosting): Combine **homogeneous** models (same type, e.g., all decision trees)
- **New methods** (Blending, Stacking, MoE): Combine **heterogeneous** models (different types, e.g., logistic regression + random forest + KNN)

Key insight: A **meta-model** learns how to best combine predictions from diverse base models!

1.1 Why This Matters

Key Information

Practical Relevance:

- In data science projects, you often try multiple models (logistic regression, SVM, random forest, boosting, etc.)
- Instead of choosing the "best" one, why not combine them?
- This is extremely valuable for final projects and Kaggle competitions
- **Modern LLMs** (like DeepSeek, Mistral) use Mixture of Experts for efficiency—this architecture is revolutionizing AI!

1.2 Review: What We've Learned

Table 1: *Ensemble Methods Review*

Method	Base Models	Training	Aggregation	Goal
Bagging	Homogeneous (trees)	Parallel	Average/Vote	Reduce variance
Random Forest	Homogeneous (trees)	Parallel	Average/Vote	Reduce variance
Boosting	Homogeneous (stumps)	Sequential	Weighted sum	Reduce bias
Blending	**Heterogeneous**	Parallel	**Meta-model**	Best of all
Stacking	**Heterogeneous**	Parallel + CV	**Meta-model**	Best of all

2 Blending: Simple Heterogeneous Ensembles

Definition: Blending

Blending is an ensemble method that:

1. Trains multiple **different** base models on training data
2. Gets predictions from these models on a held-out validation set
3. Trains a **meta-model** to learn how to combine these predictions

The meta-model learns "when to trust which model."

2.1 The Blending Architecture

Data → **Base Models** (Logistic, RF, KNN, ...) → **Predictions** → **Meta-Model** → **Final Prediction**

2.2 Step-by-Step Blending Process

Step 1: Split the Data

Split your data into four parts:

- **Training set**: To train base models
- **Validation set**: To generate predictions for meta-model training
- **Hold-out set**: To validate the meta-model
- **Test set**: Final evaluation (untouched until the end!)

Step 2: Train Base Models

Train multiple different models on the training set:

- Model 1: Logistic Regression
- Model 2: Random Forest
- Model 3: Gradient Boosting
- Model 4: KNN
- ... (as many as you want!)

Step 3: Generate Validation Predictions

Use each trained base model to predict on the validation set. This gives you:

- $\hat{y}_1$: Predictions from Model 1
- $\hat{y}_2$: Predictions from Model 2
- ... and so on

Step 4: Create Meta-Model Training Data

Combine the predictions into a new dataset:

$$X_{meta} = [\hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4, (\text{optionally: original X})]$$

$$y_{meta} = \text{true labels from validation set}$$

Step 5: Train the Meta-Model

Train a simple model (often linear regression or logistic regression) on this meta-dataset:

$$\text{Final prediction} = f_{meta}(X_{meta})$$

Step 6: Evaluate on Hold-out and Test

Use the same process for hold-out (to tune meta-model) and test (final evaluation).

2.3 Why Use a Simple Meta-Model?

Key Information

Recommendation: Keep the Meta-Model Simple!

A simple meta-model (like linear regression) has advantages:

- **Interpretability**: Coefficients tell you which base model is most trusted
- **Avoids overfitting**: Complex meta-models can overfit to validation predictions
- **Fast to train**: Meta-model training should be quick

For example, if the meta-model learns:

$$\hat{y}_{final} = 0.5 \cdot \hat{y}_{RF} + 0.3 \cdot \hat{y}_{GB} + 0.2 \cdot \hat{y}_{LR}$$

You know Random Forest contributes most to the final prediction!

2.4 The Passthrough Option

Definition: Passthrough

Passthrough means including the original features X along with the base model predictions when training the meta-model.

Without passthrough: $X_{meta} = [\hat{y}_1, \hat{y}_2, ..., \hat{y}_K]$

With passthrough: $X_{meta} = [X, \hat{y}_1, \hat{y}_2, ..., \hat{y}_K]$

Why use passthrough?

- The meta-model can learn "when input looks like THIS, trust Model 2 more"
- Base models might miss some signal that the meta-model can capture
- Generally improves performance

Warning

In sklearn, `passthrough=False` by default. The instructor recommends setting `passthrough=True` for better results!

3 Stacking: Blending with Cross-Validation

Definition: Stacking

Stacking (Stacked Generalization) is similar to blending but uses **cross-validation** instead of a fixed validation split. This makes better use of data!

3.1 The Problem with Blending

Blending requires:

- Training set (for base models)
- Validation set (for meta-model training)
- Hold-out set (for meta-model validation)
- Test set (for final evaluation)

That's 4 separate datasets! With limited data, this is wasteful.

3.2 Stacking Solution: Out-of-Fold Predictions

Stacking uses K-fold cross-validation to generate "out-of-fold" predictions:

1. Split training data into K folds (e.g., K=5)
2. For each fold:
   - Train base models on K-1 folds
   - Predict on the held-out fold
3. After K iterations, every sample has a prediction from a model that **didn't see it during training**
4. These predictions become the meta-model's training data

Example: 3-Fold Stacking Example

Data: 900 samples, 3 folds of 300 each

Round 1:

- Train on Folds 2+3 (600 samples)
- Predict on Fold 1 (300 samples) $\rightarrow \hat{y}_1^{(1)}$

Round 2:

- Train on Folds 1+3 (600 samples)
- Predict on Fold 2 (300 samples) $\rightarrow \hat{y}_1^{(2)}$

Round 3:

- Train on Folds 1+2 (600 samples)
- Predict on Fold 3 (300 samples) $\rightarrow \hat{y}_1^{(3)}$

Result: 900 out-of-fold predictions for Model 1!

Repeat for all base models. Then train meta-model on these predictions.

3.3 Stacking vs. Blending

Table 2: *Stacking vs. Blending Comparison*

Aspect	**Blending**	**Stacking**
Data efficiency	Lower	Higher
Implementation	Simpler	More complex
Computation	Faster	Slower (K-fold)
Risk of overfitting	Higher	Lower
Recommended for	Large datasets	Smaller datasets

4 Stacking in Python

```python
from sklearn.ensemble import StackingClassifier, StackingRegressor
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Define base models (heterogeneous!)
estimators = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('gb', GradientBoostingClassifier(n_estimators=100, random_state=42)),
    ('knn', KNeighborsClassifier(n_neighbors=5))
]

# Define meta-model (keep it simple!)
meta_model = LogisticRegression()

# Create stacking classifier
stacking_clf = StackingClassifier(
    estimators=estimators,
    final_estimator=meta_model,
    cv=5,                    # 5-fold cross-validation
    passthrough=True,        # Include original features!
    stack_method='predict_proba'  # Use probabilities, not labels
)

# Train
stacking_clf.fit(X_train, y_train)

# Evaluate
print(f"Stacking Train Accuracy: {stacking_clf.score(X_train, y_train):.4f}")
print(f"Stacking Test Accuracy: {stacking_clf.score(X_test, y_test):.4f}")
```

4.1 Important: Use Probabilities, Not Labels

Warning

When stacking classifiers, use `stack_method='predict_proba'` instead of hard labels!

Why?

- Labels are discrete (0 or 1)—limited information
- Probabilities are continuous (0.0 to 1.0)—much richer signal
- Meta-model can learn "Model A is confident, Model B is uncertain"

5 Mixture of Experts (MoE)

Key Summary

Mixture of Experts takes a fundamentally different approach:
Instead of combining all models' outputs, MoE **selects which expert(s) to use** based on the input!
This is the architecture behind modern efficient LLMs like DeepSeek and Mistral.

5.1 The Core Idea

In previous ensembles, ALL models contribute to every prediction:

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K} f_k(x) \quad \text{(everyone votes)}$$

In MoE, a **gating network** decides which experts to activate:

$$\hat{y} = \sum_{k=1}^{K} g_k(x) \cdot f_k(x) \quad \text{(specialists handle their domain)}$$

where $g_k(x)$ is the "gate" for expert k, and gates sum to 1: $\sum_k g_k(x) = 1$

5.2 Components of MoE

Definition: Mixture of Experts Components

1. **Experts** (f_k): Individual models that specialize in different regions of the input space
2. **Gating Network** (g): A model that looks at the input x and outputs weights for each expert
3. **Combination**: Final prediction is a weighted combination based on gates

Example: Medical Diagnosis MoE

Experts:

- Expert 1: Radiologist (specializes in imaging)
- Expert 2: Pathologist (specializes in lab tests)
- Expert 3: General Practitioner (handles common cases)

Gating Network: Looks at patient symptoms and test results

For a patient with MRI scan:

- Gate outputs: $g_1 = 0.7$, $g_2 = 0.2$, $g_3 = 0.1$
- Radiologist's opinion is weighted most heavily!

For a patient with blood work:

- Gate outputs: $g_1 = 0.1$, $g_2 = 0.8$, $g_3 = 0.1$
- Pathologist's opinion dominates!

5.3 The Gating Network

The gating network uses **softmax** to ensure gates sum to 1:

$$g_k(x) = \frac{\exp(\alpha_k^T x)}{\sum_{j=1}^{K} \exp(\alpha_j^T x)}$$

where α_k are learnable parameters for expert k.

Key Information

This is exactly like **multinomial logistic regression** (softmax)! The gating network is essentially classifying "which expert should handle this input."

5.4 Training MoE: The Challenge

Important: Expert Collapse Problem

A naive loss function can cause **expert collapse**:

- One expert becomes dominant
- Other experts never get trained
- System degenerates to single-model performance

This is what happened with early implementations (like Mistral's initial attempts).

Solution: Use a loss function that gives each expert its own error signal, not just the global error.

5.4.1 Bad Loss (Causes Collapse)

$$L = \sum_n (y_n - \sum_k g_k(x_n) \cdot f_k(x_n))^2$$

Problem: All experts see the same global error. They all try to do the same thing and collapse.

5.4.2 Good Loss (Prevents Collapse)

$$L = \sum_n \sum_k g_k(x_n) \cdot (y_n - f_k(x_n))^2$$

Benefit: Each expert gets weighted error specific to its domain. Experts specialize!

5.5 Why MoE Matters for LLMs

Key Information

Sparse MoE in Large Language Models:

Modern LLMs like DeepSeek use MoE to be more efficient:

- Model has billions of parameters spread across many experts
- For each input, only a **few experts** are activated (sparse activation)

- This gives the **capacity** of a huge model with the **computation** of a small one

Example: A model with 100 experts but only top-2 activated per token can have 10x the parameters with only 2x the compute!

6 Practical Guidelines

6.1 When to Use Each Method

Table 3: *Choosing an Ensemble Method*

Situation	Recommended Method
Quick baseline, plenty of data	Random Forest or Gradient Boosting
Want to squeeze last 1% accuracy	Stacking with diverse models
Data has distinct regions/modes	Mixture of Experts
Need interpretability	Stacking with linear meta-model
Limited computational resources	Blending (simpler than stacking)
Building a large-scale system	Sparse Mixture of Experts

6.2 Best Practices

1. **Diversity is key**: Use models with different "philosophies"
   - Linear model (captures global trends)
   - Tree model (captures interactions)
   - KNN (captures local patterns)
   - Boosting (captures complex patterns)
2. **Watch for data leakage**: Never use test data during stacking/blending
3. **Keep meta-model simple**: Complex meta-models tend to overfit
4. **Use probabilities for classification**: More informative than labels
5. **Enable passthrough**: Let meta-model see original features
6. **Validate properly**: Use hold-out set for meta-model tuning

7 Course Summary: The ML Toolkit

Key Summary

What You've Learned in CS109A:

Regression:

- Linear Regression, Polynomial Regression
- Regularization (Ridge, Lasso)

Classification:

- Logistic Regression
- K-Nearest Neighbors

Probabilistic Methods:

- Bayesian Inference
- MCMC Sampling

Tree-Based Methods:

- Decision Trees (Classification and Regression)
- Bagging and Random Forests
- Gradient Boosting and AdaBoost

Advanced Ensembles:

- Blending and Stacking
- Mixture of Experts

Important Concepts Throughout:

- Bias-Variance Tradeoff
- Cross-Validation
- Feature Engineering
- Model Selection and Evaluation

Key Information

Final Thoughts from the Instructors:

1. **Use office hours!** You learn more in 30 minutes with a TA than hours struggling alone.
2. **Don't over-rely on LLMs**: They're good for small questions, but can lead you astray for learning core concepts.
3. **Start simple**: Even with all these methods, linear/logistic regression should be your first attempt. Understand the basics before reaching for complex tools.
4. **For tabular data**: Random Forest and Gradient Boosting often beat deep learning!
5. **Keep learning**: This course is just the beginning. There's much more in CS109B and beyond.

8 Practice Questions

1. **Conceptual**: Explain the difference between homogeneous and heterogeneous ensembles. Give an example of each.
2. **Blending**: Why do we need a separate validation set for blending? What would happen if we used the same data to train base models and generate meta-model training data?
3. **Stacking**: Explain what "out-of-fold predictions" are and why they prevent overfitting in the meta-model.
4. **Passthrough**: What is the passthrough option in stacking? Why is it recommended to enable it?
5. **MoE Gating**: In a Mixture of Experts model, what does the gating network do? What function ensures the gate outputs sum to 1?
6. **Expert Collapse**: What is expert collapse in MoE, and why does a naive loss function cause it?
7. **Practical**: You're competing in a Kaggle competition. You've trained:
   - Logistic Regression: 82% accuracy
   - Random Forest: 85% accuracy
   - Gradient Boosting: 86% accuracy
   - KNN: 80% accuracy

   How would you use stacking to potentially improve beyond 86%? Write the sklearn code.
8. **LLM Application**: Why are Large Language Models using Mixture of Experts architectures? What efficiency benefit does sparse MoE provide?