Model Development & Training

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Today's Agenda

- Model selection strategies
- Training best practices
- Hyperparameter optimization
- AutoML with AutoGluon
- Model checkpointing
- Transfer learning
- Training pipelines

Model Development Lifecycle

It's not just `model.fit()`. It's a cycle.

graph LR A[Data Prep] --> B[Feat Eng]; B --> C[Model Selection]; C --> D[Training]; D --> E[Evaluation]; E --> F{Good Enough?}; F -- No --> C; F -- Yes --> G[Deployment]; E --> H[Error Analysis]; H --> A;

Iterative Process:

1. Start simple (Baseline).
2. Analyze errors.
3. Add complexity (New features, complex models).

Model Selection Strategy

Don't start with a Transformer. Start with a baseline.

Data Type	Baseline (Fast, Simple)	Advanced (SOTA, Heavy)
Tabular	Logistic Regression, Decision Tree	XGBoost, LightGBM, TabNet
Image	ResNet-18	EfficientNet, ViT (Vision Transformer)
Text	TF-IDF + Naive Bayes	BERT, RoBERTa, GPT
Time Series	ARIMA, Linear Regression	LSTM, Transformer

Why Baselines?

- Debug pipeline bugs quickly.
- Establish a "floor" for performance.
- If deep learning only gives +1% over Logistic Regression, is it worth the cost?

Bias vs. Variance Trade-off

Bias (Underfitting): Model is too simple to capture patterns.
Variance (Overfitting): Model memorizes noise in training data.

graph TD A[Total Error] --> B[Bias^2]; A --> C[Variance]; A --> D[Irreducible Error];
Goal: Sweet spot where Total Error is minimized.

Fixing Overfitting:

- More data
- Regularization (L1/L2, Dropout)
- Simpler model

Fixing Underfitting:

- More features

Data Splitting Strategies

1. **Hold-out Set**:
   - Train (60%), Validation (20%), Test (20%).
   - **Risk**: Validation set might be lucky/unlucky.
2. **K-Fold Cross-Validation**:
   - Robust estimate of performance.
   - Train K times on K different splits.

gantt title 5-Fold Cross Validation dateFormat X axisFormat %s section Fold 1 Test : 0, 20 Train : 20, 100 section Fold 2 Train : 0, 20 Test : 20, 40 Train : 40, 100 section Fold 3 Train : 0, 40 Test : 40, 60 Train : 60, 100

Hyperparameter Optimization (HPO)

Parameters: Learned from data (Weights, Biases).

Hyperparameters: Set *before* training (Learning Rate, Batch Size, Depth).

Search Strategies:

1. **Grid Search**: Try *every* combination.
    - Safe but exponentially expensive ($O(N^D)$).
2. **Random Search**: Randomly sample configurations.
    - Surprisingly effective.
3. **Bayesian Optimization (Optuna)**: Smart search.
    - "Given that `lr=0.1` was bad, don't try `lr=0.2`, try `lr=0.01`."

Bayesian Optimization Visualized

How it works:

1. Build a probability model of the objective function.
2. Choose next hyperparameter to query (Exploration vs Exploitation).
3. Update model.

Optuna Code:

```
def objective(trial):
    # Suggest params
    lr = trial.suggest_loguniform('lr', 1e-5, 1e-1)
    depth = trial.suggest_int('depth', 3, 10)

    model = Train(lr, depth)
    return model.val_accuracy

study.optimize(objective, n_trials=100)
```

AutoML: Automated Machine Learning

Philosophy: "I don't care which model, just give me the best one."

AutoGluon (Amazon) creates a stacked ensemble of models.

graph TD Data --> M1[Random Forest]; Data --> M2[CatBoost]; Data --> M3[Neural Net]; M1 --> L2[Weighted Ensemble]; M2 --> L2; M3 --> L2; L2 --> Prediction;

Pros: SOTA performance with 3 lines of code.

Cons: Slow training, heavy inference, hard to interpret.

Training Pipelines

Spaghetti code in notebooks is the enemy of reproducibility.

Pipeline: A reproducible recipe.

`Raw Data` -> `Imputer` -> `Scaler` -> `Encoder` -> `Model`

scikit-learn Pipeline:

```
pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=50)),
    ('clf', RandomForestClassifier())
])
pipe.fit(X_train, y_train)
```

Benefits: No data leakage! Transforms are fit *only* on train splits during CV.

Checkpointing & Early Stopping

The "Epoch" Dilemma:

- Train too little -> Underfit.
- Train too much -> Overfit.

Early Stopping:

- Monitor Validation Loss.
- If it stops decreasing for `patience` epochs -> **STOP**.

Checkpointing:

- Save the model weights *every time* validation loss improves.
- Restore the *best* version at the end.

Transfer Learning: Theory

Don't reinvent the wheel.
Someone (Google/Meta) spent $10M to train a model on ImageNet (14M images). It learned to see edges, textures, shapes.

Strategies:

1. **Feature Extraction**: Freeze backbone, train only the head (classifier).
    - Fast, low data requirement.
2. **Fine-Tuning**: Unfreeze backbone (or parts of it) and train with low learning rate.
    - Slower, needs more data, higher accuracy.

graph TD A[Pre-trained Backbone] --> B[New Head]; subgraph "Feature Extraction" A:::frozen B:::trainable end classDef frozen fill:#eee,stroke:#333,stroke-dasharray: 5 5; classDef trainable fill:#bbf,stroke:#333;

Experiment Tracking

Problem: "I trained 50 models. Which one had `lr=0.001` and `dropout=0.5`?"

Solution: Use a tracker (Weights & Biases, MLflow).

What to track:

- **Config**: Hyperparameters (yaml/json).
- **Metrics**: Loss, Accuracy, F1 (charts).
- **Artifacts**: Model weights (`.pt`), dataset versions.
- **System**: GPU usage, memory.

Best Practices Summary

1. **Baseline First**: Always beat a dummy classifier.
2. **Leakage Free**: Use Pipelines.
3. **Track Everything**: Use W&B/MLflow.
4. **Save Often**: Checkpoints are life-savers.
5. **Be Lazy**: Use Transfer Learning and AutoML where possible.

Lab Preview

Hands-on exercises:

1. **Manual**: Compare SVM vs Random Forest with Cross-Validation.
2. **Automated**: Use AutoGluon to beat your manual models.
3. **Optimization**: Use Optuna to tune the Random Forest.
4. **Tracking**: Log everything to W&B.

Let's code!