# Model Development & Training

## Lab Session

**CS 203: Software Tools and Techniques for AI**

Prof. Nipun Batra, IIT Gandhinagar

## Lab Objectives

1. Compare multiple models with cross-validation
2. Perform hyperparameter tuning with Optuna
3. Use AutoGluon for automated ML
4. Build training pipelines
5. Implement transfer learning
6. Track experiments with Weights & Biases
7. **(Bonus) Fine-tune an LLM with PEFT**

## Setup

```
pip install scikit-learn pandas numpy
pip install optuna autogluon
pip install torch torchvision
pip install transformers datasets peft accelerate
pip install wandb pytorch-lightning
```

## Exercise 1: Model Selection

**Task**: Compare different models on a classification task

**Dataset**: Wine Quality

- Features: chemical properties
- Target: wine quality (0-10)

**Models to compare**:

- LogisticRegression
- RandomForest
- GradientBoosting
- SVC

# Exercise 1: Load Data

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load wine quality dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
df = pd.read_csv(url, sep=';')

# Convert to binary classification
df['quality_binary'] = (df['quality'] >= 6).astype(int)

X = df.drop(['quality', 'quality_binary'], axis=1)
y = df['quality_binary']

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Train: {len(X_train)}, Test: {len(X_test)}")
print(f"Class distribution: {y_train.value_counts()}")
```

# Exercise 1: Compare Models

```python
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

models = {
    'LogisticRegression': Pipeline([
        ('scaler', StandardScaler()),
        ('model', LogisticRegression(max_iter=1000))
    ]),
    'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
    'GradientBoosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'SVC': Pipeline([
        ('scaler', StandardScaler()),
        ('model', SVC())
    ])
}
```

## Exercise 1: Cross-Validation

```python
results = {}

for name, model in models.items():
    scores = cross_val_score(
        model, X_train, y_train,
        cv=5, scoring='f1_weighted'
    )
    results[name] = {
        'mean': scores.mean(),
        'std': scores.std(),
        'scores': scores
    }
    print(f"{name:20} F1: {scores.mean():.3f} (+/- {scores.std():.3f})")

# Find best model
best_model_name = max(results, key=lambda k: results[k]['mean'])
print(f"\nBest model: {best_model_name}")
```

## Exercise 1: Train and Evaluate

```
from sklearn.metrics import classification_report, confusion_matrix

# Train best model on full training set
best_model = models[best_model_name]
best_model.fit(X_train, y_train)

# Evaluate on test set
y_pred = best_model.predict(X_test)

print("\nTest Set Performance:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

## Exercise 2: Hyperparameter Tuning with Optuna

**Task**: Optimize RandomForest hyperparameters

**Hyperparameters to tune**:

- n_estimators
- max_depth
- min_samples_split
- min_samples_leaf

## Exercise 2: Define Objective

```python
import optuna
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

def objective(trial):
    # Suggest hyperparameters
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 50, 300),
        'max_depth': trial.suggest_int('max_depth', 3, 20),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
        'random_state': 42
    }

    model = RandomForestClassifier(**params)
    score = cross_val_score(
        model, X_train, y_train,
        cv=5, scoring='f1_weighted'
    ).mean()

    return score
```

## Exercise 2: Run Optimization

```
# Create study
study = optuna.create_study(
    direction='maximize',
    study_name='rf_optimization'
)

# Optimize
study.optimize(objective, n_trials=100)

# Results
print(f"Best value: {study.best_value:.3f}")
print(f"Best params: {study.best_params}")

# Train with best params
best_rf = RandomForestClassifier(**study.best_params)
best_rf.fit(X_train, y_train)

test_score = f1_score(y_test, best_rf.predict(X_test), average='weighted')
print(f"Test F1 score: {test_score:.3f}")
```

## Exercise 2: Visualize Optimization

```
from optuna.visualization import plot_optimization_history, plot_param_importances

# Optimization history
fig1 = plot_optimization_history(study)
fig1.show()

# Parameter importances
fig2 = plot_param_importances(study)
fig2.show()

# Plot parallel coordinate
fig3 = optuna.visualization.plot_parallel_coordinate(study)
fig3.show()
```

## Exercise 3: AutoML with AutoGluon

**Task**: Use AutoGluon for automatic model selection and ensembling

**Dataset**: Titanic survival prediction

## Exercise 3: Load Titanic Data

```
import pandas as pd
from autogluon.tabular import TabularPredictor

# Load Titanic dataset
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
df = pd.read_csv(url)

# Basic preprocessing
df = df.drop(['Name', 'Ticket', 'Cabin'], axis=1)

# Split
train_data = df.sample(frac=0.8, random_state=42)
test_data = df.drop(train_data.index)

print(f"Train: {len(train_data)}, Test: {len(test_data)}")
```

## Exercise 3: Train AutoGluon

```python
# Initialize predictor
predictor = TabularPredictor(
    label='Survived',
    eval_metric='f1',
    path='./ag_titanic/'
)

# Train
predictor.fit(
    train_data,
    time_limit=300,  # 5 minutes
    presets='medium_quality'
)

# View leaderboard
leaderboard = predictor.leaderboard(test_data, silent=True)
print(leaderboard)
```

## Exercise 3: Predictions and Insights

```
# Make predictions
predictions = predictor.predict(test_data)
pred_probs = predictor.predict_proba(test_data)

# Evaluate
from sklearn.metrics import classification_report
print(classification_report(test_data['Survived'], predictions))

# Feature importance
importance = predictor.feature_importance(train_data)
print("\nTop 10 Features:")
print(importance.head(10))

# Explain a prediction
sample_idx = 0
explanation = predictor.explain(test_data.iloc[[sample_idx]])
```

## Exercise 3: Compare Presets

```
presets = ['medium_quality', 'best_quality', 'optimize_for_deployment']
results = {}

for preset in presets:
    predictor = TabularPredictor(
        label='Survived',
        eval_metric='f1',
        path=f'./ag_titanic_{preset}/'
    )

    predictor.fit(train_data, time_limit=300, presets=preset)

    # Evaluate
    perf = predictor.evaluate(test_data)
    results[preset] = perf

print("Performance by preset:")
for preset, perf in results.items():
    print(f"{preset:25} F1: {perf['f1']:.3f}")
```

## Exercise 4: Training Pipeline

**Task**: Build end-to-end scikit-learn pipeline

**Components**:

- Feature scaling
- PCA dimensionality reduction
- Model training

## Exercise 4: Create Pipeline

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import GradientBoostingClassifier

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),  # Keep 95% variance
    ('classifier', GradientBoostingClassifier(random_state=42))
])

# Fit pipeline
pipeline.fit(X_train, y_train)

# Predict
y_pred = pipeline.predict(X_test)

from sklearn.metrics import accuracy_score, f1_score
print(f"Accuracy: {accuracy_score(y_test, y_pred):.3f}")
print(f"F1 Score: {f1_score(y_test, y_pred, average='weighted'):.3f}")
```

## Exercise 4: Tune Pipeline

```python
from sklearn.model_selection import GridSearchCV

param_grid = {
    'pca__n_components': [0.90, 0.95, 0.99],
    'classifier__n_estimators': [50, 100, 200],
    'classifier__learning_rate': [0.01, 0.1, 0.3],
    'classifier__max_depth': [3, 5, 7]
}

grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=5,
    scoring='f1_weighted',
    n_jobs=-1,
    verbose=2
)

grid_search.fit(X_train, y_train)

print(f"Best params: {grid_search.best_params_}")
print(f"Best CV score: {grid_search.best_score_:.3f}")

# Test set performance
test_score = grid_search.score(X_test, y_test)
print(f"Test score: {test_score:.3f}")
```

## Exercise 4: Save Pipeline

```python
import joblib

# Save fitted pipeline
joblib.dump(grid_search.best_estimator_, 'best_pipeline.pkl')

# Load and use
loaded_pipeline = joblib.load('best_pipeline.pkl')
new_predictions = loaded_pipeline.predict(X_test)
```

## Exercise 5: Transfer Learning with PyTorch

**Task**: Fine-tune pre-trained ResNet for image classification

**Dataset**: CIFAR-10

## Exercise 5: Setup

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision import models

# Device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Data transforms
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

## Exercise 5: Load CIFAR-10

```
# Download and load data
trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform_train
)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=128, shuffle=True, num_workers=2
)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform_test
)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=128, shuffle=False, num_workers=2
)

classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')
```

## Exercise 5: Load Pre-trained Model

```python
# Load ResNet18 pre-trained on ImageNet
model = models.resnet18(pretrained=True)

# Freeze all layers
for param in model.parameters():
    param.requires_grad = False

# Replace final layer for 10 classes
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)

model = model.to(device)

# Only train the final layer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
```

## Exercise 5: Training Function

```
def train_epoch(model, trainloader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in trainloader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    return running_loss / len(trainloader), 100. * correct / total
```

## Exercise 5: Evaluation Function

```python
def evaluate(model, testloader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    return running_loss / len(testloader), 100. * correct / total
```

## Exercise 5: Train Model

```
num_epochs = 10
best_acc = 0

for epoch in range(num_epochs):
    train_loss, train_acc = train_epoch(
        model, trainloader, criterion, optimizer, device
    )
    val_loss, val_acc = evaluate(model, testloader, criterion, device)

    print(f"Epoch {epoch+1}/{num_epochs}")
    print(f"  Train Loss: {train_loss:.3f}, Acc: {train_acc:.2f}%")
    print(f"  Val Loss: {val_loss:.3f}, Acc: {val_acc:.2f}%")

    # Save best model
    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(model.state_dict(), 'best_resnet_cifar10.pt')
        print(f"  Saved best model (acc: {best_acc:.2f}%)")
```

## Exercise 5: Fine-tuning

```python
# Unfreeze layer4 for fine-tuning
for param in model.layer4.parameters():
    param.requires_grad = True

# Use different learning rates
optimizer = optim.Adam([
    {'params': model.layer4.parameters(), 'lr': 1e-4},
    {'params': model.fc.parameters(), 'lr': 1e-3}
])

# Continue training
for epoch in range(5):
    train_loss, train_acc = train_epoch(
        model, trainloader, criterion, optimizer, device
    )
    val_loss, val_acc = evaluate(model, testloader, criterion, device)

    print(f"Fine-tune Epoch {epoch+1}/5")
    print(f"  Train Acc: {train_acc:.2f}%, Val Acc: {val_acc:.2f}%")
```

## Exercise 6: Experiment Tracking with W&B

**Task**: Track experiments with Weights & Biases

## Exercise 6: Setup W&B

```python
import wandb

# Login (first time only)
# wandb.login()

# Initialize run
wandb.init(
    project='wine-quality-classification',
    config={
        'learning_rate': 0.001,
        'epochs': 10,
        'batch_size': 32,
        'model': 'RandomForest',
        'n_estimators': 100
    }
)

config = wandb.config
```

# Exercise 6: Log Metrics

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score

# Train model
model = RandomForestClassifier(
    n_estimators=config.n_estimators,
    random_state=42
)
model.fit(X_train, y_train)

# Evaluate
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

# Log metrics
wandb.log({
    'train_accuracy': accuracy_score(y_train, train_pred),
    'test_accuracy': accuracy_score(y_test, test_pred),
    'train_f1': f1_score(y_train, train_pred, average='weighted'),
    'test_f1': f1_score(y_test, test_pred, average='weighted')
})
```

## Exercise 6: Log Feature Importance

```python
import matplotlib.pyplot as plt

# Get feature importance
importance = model.feature_importances_
feature_names = X_train.columns

# Create plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(feature_names, importance)
ax.set_xlabel('Importance')
ax.set_title('Feature Importance')

# Log to W&B
wandb.log({'feature_importance': wandb.Image(fig)})
plt.close()
```

## Exercise 6: Log Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Create confusion matrix
cm = confusion_matrix(y_test, test_pred)

# Plot
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
ax.set_title('Confusion Matrix')

# Log to W&B
wandb.log({'confusion_matrix': wandb.Image(fig)})
plt.close()

# Finish run
wandb.finish()
```

## Exercise 7: Hyperparameter Sweep with W&B

**Task**: Systematic hyperparameter search with W&B

## Exercise 7: Define Sweep Config

```python
sweep_config = {
    'method': 'bayes',  # or 'grid', 'random'
    'metric': {
        'name': 'test_f1',
        'goal': 'maximize'
    },
    'parameters': {
        'n_estimators': {
            'values': [50, 100, 200, 300]
        },
        'max_depth': {
            'values': [5, 10, 15, 20, None]
        },
        'min_samples_split': {
            'min': 2,
            'max': 20
        },
        'min_samples_leaf': {
            'min': 1,
            'max': 10
        }
    }
}

sweep_id = wandb.sweep(sweep_config, project='wine-quality-sweep')
```

## Exercise 7: Define Training Function

```
def train_sweep():
    # Initialize run
    with wandb.init() as run:
        config = wandb.config

        # Train model
        model = RandomForestClassifier(
            n_estimators=config.n_estimators,
            max_depth=config.max_depth,
            min_samples_split=config.min_samples_split,
            min_samples_leaf=config.min_samples_leaf,
            random_state=42
        )
        model.fit(X_train, y_train)

        # Evaluate
        test_pred = model.predict(X_test)
        test_f1 = f1_score(y_test, test_pred, average='weighted')

        # Log
        wandb.log({'test_f1': test_f1})

# Run sweep
wandb.agent(sweep_id, train_sweep, count=50)
```

## Challenge: Fine-tuning LLM with PEFT (Advanced)

**Task**: Fine-tune a tiny BERT model for sentiment analysis using LoRA.

**Steps**:

1. Load dataset (e.g., `imdb` or `rotten_tomatoes`).
2. Load pre-trained model (`distilbert-base-uncased`).
3. Configure LoRA (`LoraConfig`).
4. Wrap model (`get_peft_model`).
5. Train for 1 epoch using `Trainer`.

**Hint**: Use `peft` and `transformers` libraries.

## Common Issues

**Overfitting**:

- Model too complex
- Training too long
- Solution: regularization, early stopping

**Underfitting**:

- Model too simple
- Not enough features
- Solution: more complex model, feature engineering

**Data leakage**:

- Scaling before splitting
- Solution: use pipelines

## Key Takeaways

1. Compare multiple models before committing
2. Hyperparameter tuning can improve performance 10-20%
3. AutoML provides strong baselines
4. Transfer learning saves time and improves accuracy
5. Track all experiments for reproducibility
6. Use pipelines for clean workflows
7. W&B makes experiment tracking easy

## Additional Resources

**Documentation**:

- Scikit-learn: scikit-learn.org
- Optuna: optuna.readthedocs.io
- AutoGluon: auto.gluon.ai
- PyTorch: pytorch.org
- Weights & Biases: docs.wandb.ai
- Hugging Face PEFT: huggingface.co/docs/peft

**Tutorials**:

- AutoGluon Quick Start: auto.gluon.ai/tutorials
- Transfer Learning: pytorch.org/tutorials
- Optuna Examples: github.com/optuna/optuna-examples

## Next Steps

Continue experimenting:

- Try different datasets
- Explore ensemble methods
- Implement custom models
- Compare AutoML tools
- Build production pipelines
- Deploy models (next week!)