Active Learning Lab

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Overview

By the end of this lab, you will:

- Implement uncertainty sampling from scratch
- Build complete active learning loop
- Compare active vs random sampling
- Use modAL library
- Apply active learning to real datasets
- Visualize and analyze results

Structure:

- Part 1: Basic Implementation (45 min)
- Part 2: Query Strategies (45 min)
- Part 3: Experiments & Analysis (60 min)

Setup (10 minutes)

Install packages:

```
pip install numpy scikit-learn matplotlib pandas modAL-python
```

Verify installation:

```python
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
from modAL.models import ActiveLearner

print("Setup complete!")
```

Exercise 1.1: Generate Toy Dataset (10 min)

Task: Create synthetic classification dataset

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import numpy as np

# Generate binary classification dataset
X, y = make_classification(
    n_samples=1000,
    n_features=20,
    n_informative=15,
    n_redundant=5,
    n_classes=2,
    random_state=42
)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"Training samples: {len(X_train)}")
print(f"Test samples: {len(X_test)}")
print(f"Class distribution: {np.bincount(y_train)}")
```

Exercise 1.2: Create Initial Split (10 min)

Task: Split training data into initial labeled set and pool

```python
def create_initial_split(X_train, y_train, n_initial=20):
    # Randomly select initial labeled examples
    n_samples = len(X_train)
    indices = np.arange(n_samples)
    np.random.shuffle(indices)

    # Split indices
    labeled_idx = indices[:n_initial]
    pool_idx = indices[n_initial:]

    # Create initial labeled set
    X_labeled = X_train[labeled_idx]
    y_labeled = y_train[labeled_idx]

    # Create unlabeled pool
    X_pool = X_train[pool_idx]
    y_pool = y_train[pool_idx]  # True labels (oracle)

    return X_labeled, y_labeled, X_pool, y_pool, pool_idx

# Create split
X_labeled, y_labeled, X_pool, y_pool, pool_idx = create_initial_split(
    X_train, y_train, n_initial=20
)

print(f"Initial labeled: {len(X_labeled)}")
print(f"Unlabeled pool: {len(X_pool)}")
```

Exercise 1.3: Implement Uncertainty Sampling (15 min)

Task: Build uncertainty sampling function

```python
def uncertainty_sampling(model, X_pool, n_samples=10):
    """
    Select samples with highest uncertainty.

    Args:
        model: Trained classifier with predict_proba
        X_pool: Unlabeled samples
        n_samples: Number of samples to query

    Returns:
        indices: Indices of selected samples
    """
    # Get prediction probabilities
    probas = model.predict_proba(X_pool)

    # Calculate uncertainty (1 - max probability)
    uncertainties = 1 - np.max(probas, axis=1)

    # Select top uncertain samples
    query_idx = np.argsort(uncertainties)[-n_samples:]

    return query_idx

# Test
model = LogisticRegression(random_state=42)
model.fit(X_labeled, y_labeled)
```

Exercise 1.4: Oracle Function (10 min)

Task: Simulate human labeler

```python
def oracle(X_query, y_pool, query_idx):
    """
    Simulate perfect oracle that returns true labels.

    Args:
        X_query: Queried samples
        y_pool: True labels of pool
        query_idx: Indices in pool

    Returns:
        labels: True labels for queried samples
    """
    return y_pool[query_idx]

def noisy_oracle(X_query, y_pool, query_idx, error_rate=0.1):
    """
    Oracle with label noise.
    """
    labels = y_pool[query_idx].copy()

    # Flip some labels
    n_errors = int(len(labels) * error_rate)
    if n_errors > 0:
        error_idx = np.random.choice(len(labels), size=n_errors, replace=False)
        labels[error_idx] = 1 - labels[error_idx]

    return labels

# Test
labels = oracle(X_pool[query_idx], y_pool, query_idx)
print(f"Oracle labels: {labels}")

noisy_labels = noisy_oracle(X_pool[query_idx], y_pool, query_idx, error_rate=0.2)
print(f"Noisy labels: {noisy_labels}")
```

Exercise 1.5: Active Learning Loop (15 min)

Task: Implement complete active learning cycle

```
def active_learning_loop(
    X_labeled, y_labeled, X_pool, y_pool,
    X_test, y_test,
    n_iterations=20,
    n_queries=10
):
    """
    Run active learning loop.

    Returns:
        accuracies: List of test accuracies at each iteration
        n_labeled: List of labeled set sizes
    """
    accuracies = []
    n_labeled = []

    model = LogisticRegression(random_state=42, max_iter=1000)

    for iteration in range(n_iterations):
        # Train model
        model.fit(X_labeled, y_labeled)

        # Evaluate
        accuracy = model.score(X_test, y_test)
        accuracies.append(accuracy)
        n_labeled.append(len(X_labeled))

        print(f"Iteration {iteration+1}: {len(X_labeled)} labels, Accuracy = {accuracy:.3f}")

        # Check if pool is empty
        if len(X_pool) == 0:
            break

        # Query samples
        n_query = min(n_queries, len(X_pool))
        query_idx = uncertainty_sampling(model, X_pool, n_samples=n_query)

        # Get labels from oracle
        new_labels = oracle(X_pool[query_idx], y_pool, query_idx)

        # Add to labeled set
        X_labeled = np.vstack([X_labeled, X_pool[query_idx]])
        y_labeled = np.hstack([y_labeled, new_labels])

        # Remove from pool
        X_pool = np.delete(X_pool, query_idx, axis=0)
        y_pool = np.delete(y_pool, query_idx, axis=0)

    return accuracies, n_labeled
```

Exercise 1.6: Run Active Learning (10 min)

Task: Execute active learning loop

```python
# Reset initial split
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

# Run active learning
active_accuracies, active_n_labeled = active_learning_loop(
    X_labeled, y_labeled, X_pool, y_pool,
    X_test, y_test,
    n_iterations=20,
    n_queries=10
)

print(f"\nFinal accuracy: {active_accuracies[-1]:.3f}")
print(f"Total labels used: {active_n_labeled[-1]}")
```

Expected output:

Exercise 2.1: Random Sampling Baseline (15 min)

Task: Implement random sampling for comparison

```python
def random_sampling_loop(
    X_labeled, y_labeled, X_pool, y_pool,
    X_test, y_test,
    n_iterations=20,
    n_queries=10
):
    """
    Random sampling baseline (no active learning).
    """
    accuracies = []
    n_labeled = []

    model = LogisticRegression(random_state=42, max_iter=1000)

    for iteration in range(n_iterations):
        # Train model
        model.fit(X_labeled, y_labeled)

        # Evaluate
        accuracy = model.score(X_test, y_test)
        accuracies.append(accuracy)
        n_labeled.append(len(X_labeled))

        if len(X_pool) == 0:
            break

        # Random selection (key difference)
        n_query = min(n_queries, len(X_pool))
        query_idx = np.random.choice(len(X_pool), size=n_query, replace=False)

        # Get labels and update
        new_labels = oracle(X_pool[query_idx], y_pool, query_idx)
        X_labeled = np.vstack([X_labeled, X_pool[query_idx]])
        y_labeled = np.hstack([y_labeled, new_labels])
        X_pool = np.delete(X_pool, query_idx, axis=0)
        y_pool = np.delete(y_pool, query_idx, axis=0)

    return accuracies, n_labeled
```

Exercise 2.2: Compare Strategies (10 min)

Task: Run both strategies and compare

```
# Reset for fair comparison
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

# Run random sampling
random_accuracies, random_n_labeled = random_sampling_loop(
    X_labeled.copy(), y_labeled.copy(), X_pool.copy(), y_pool.copy(),
    X_test, y_test,
    n_iterations=20,
    n_queries=10
)

# Reset and run active learning
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

active_accuracies, active_n_labeled = active_learning_loop(
    X_labeled, y_labeled, X_pool, y_pool,
    X_test, y_test,
    n_iterations=20,
    n_queries=10
)

# Compare final accuracies
print(f"\nRandom Sampling: {random_accuracies[-1]:.3f}")
print(f"Active Learning: {active_accuracies[-1]:.3f}")
print(f"Improvement: {(active_accuracies[-1] - random_accuracies[-1]):.3f}")
```

Exercise 2.3: Margin Sampling (15 min)

Task: Implement alternative uncertainty measure

```python
def margin_sampling(model, X_pool, n_samples=10):
    """
    Select samples with smallest margin between top 2 classes.
    Smaller margin = more uncertain.
    """
    probas = model.predict_proba(X_pool)

    # Sort probabilities for each sample
    sorted_probas = np.sort(probas, axis=1)

    # Margin = difference between top 2 probabilities
    margins = sorted_probas[:, -1] - sorted_probas[:, -2]

    # Select samples with smallest margins (most uncertain)
    query_idx = np.argsort(margins)[:n_samples]

    return query_idx

# Test margin sampling
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

model = LogisticRegression(random_state=42)
model.fit(X_labeled, y_labeled)

query_idx = margin_sampling(model, X_pool, n_samples=5)
probas = model.predict_proba(X_pool[query_idx])
margins = np.sort(probas, axis=1)[:, -1] - np.sort(probas, axis=1)[:, -2]

print(f"Selected margins: {margins}")
print(f"Average margin: {np.mean(margins):.3f}")
```

Exercise 2.4: Entropy Sampling (15 min)

Task: Implement entropy-based uncertainty

```python
from scipy.stats import entropy

def entropy_sampling(model, X_pool, n_samples=10):
    """
    Select samples with highest prediction entropy.
    """
    probas = model.predict_proba(X_pool)

    # Calculate entropy for each sample
    entropies = entropy(probas.T)

    # Select samples with highest entropy
    query_idx = np.argsort(entropies)[-n_samples:]

    return query_idx

# Compare all three strategies
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

model = LogisticRegression(random_state=42)
model.fit(X_labeled, y_labeled)

# Get samples from each strategy
uncertain_idx = uncertainty_sampling(model, X_pool, n_samples=10)
margin_idx = margin_sampling(model, X_pool, n_samples=10)
entropy_idx = entropy_sampling(model, X_pool, n_samples=10)

# Check overlap
overlap_um = len(set(uncertain_idx) & set(margin_idx))
overlap_ue = len(set(uncertain_idx) & set(entropy_idx))
overlap_me = len(set(margin_idx) & set(entropy_idx))

print(f"Uncertainty-Margin overlap: {overlap_um}/10")
print(f"Uncertainty-Entropy overlap: {overlap_ue}/10")
print(f"Margin-Entropy overlap: {overlap_me}/10")
```

Exercise 3.1: Visualize Learning Curves (15 min)

Task: Plot active vs random sampling

```python
import matplotlib.pyplot as plt

def plot_learning_curves(active_acc, random_acc, active_n, title="Learning Curves"):
    plt.figure(figsize=(10, 6))

    plt.plot(active_n, active_acc, 'o-', label='Active Learning', linewidth=2)
    plt.plot(active_n, random_acc, 's-', label='Random Sampling', linewidth=2)

    plt.xlabel('Number of Labeled Samples', fontsize=12)
    plt.ylabel('Test Accuracy', fontsize=12)
    plt.title(title, fontsize=14)
    plt.legend(fontsize=11)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig('learning_curves.png', dpi=300)
    plt.show()

    # Print key statistics
    idx_90 = np.argmax(np.array(active_acc) >= 0.90)
    if active_acc[idx_90] >= 0.90:
        print(f"\nActive learning reaches 90% at {active_n[idx_90]} labels")

    idx_90_random = np.argmax(np.array(random_acc) >= 0.90)
    if random_acc[idx_90_random] >= 0.90:
        print(f"Random sampling reaches 90% at {active_n[idx_90_random]} labels")
        savings = active_n[idx_90_random] - active_n[idx_90]
        print(f"Label savings: {savings} ({savings/active_n[idx_90_random]*100:.1f}%)")

plot_learning_curves(active_accuracies, random_accuracies, active_n_labeled)
```

Exercise 3.2: Analyze Selected Samples (15 min)

Task: Understand what samples are being selected

```python
def analyze_selected_samples(model, X_labeled, X_pool, y_pool, n_samples=20):
    """
    Analyze characteristics of selected samples.
    """
    query_idx = uncertainty_sampling(model, X_pool, n_samples=n_samples)

    # Get probabilities for selected samples
    probas = model.predict_proba(X_pool[query_idx])
    max_probas = np.max(probas, axis=1)

    # Get true labels
    true_labels = y_pool[query_idx]
    pred_labels = model.predict(X_pool[query_idx])

    # Calculate statistics
    accuracy = np.mean(pred_labels == true_labels)

    print(f"Selected {n_samples} samples:")
    print(f"  Average max probability: {np.mean(max_probas):.3f}")
    print(f"  Min max probability: {np.min(max_probas):.3f}")
    print(f"  Max max probability: {np.max(max_probas):.3f}")
    print(f"  Model accuracy on selected: {accuracy:.3f}")
    print(f"  Class distribution: {np.bincount(true_labels)}")

    # Visualize uncertainty distribution
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.hist(max_probas, bins=20, edgecolor='black')
    plt.xlabel('Max Probability')
    plt.ylabel('Count')
    plt.title('Selected Samples Certainty')

    plt.subplot(1, 2, 2)
    all_probas = model.predict_proba(X_pool)
    all_max_probas = np.max(all_probas, axis=1)
    plt.hist(all_max_probas, bins=50, alpha=0.5, label='All pool')
    plt.hist(max_probas, bins=20, alpha=0.7, label='Selected', edgecolor='black')
    plt.xlabel('Max Probability')
    plt.ylabel('Count')
    plt.legend()
    plt.title('Selected vs All Pool')

    plt.tight_layout()
    plt.show()

# Run analysis
model = LogisticRegression(random_state=42)
model.fit(X_labeled, y_labeled)
analyze_selected_samples(model, X_labeled, X_pool, y_pool, n_samples=20)
```

Exercise 3.3: Cost-Effectiveness Analysis (15 min)

Task: Calculate labeling cost savings

```python
def cost_analysis(active_acc, random_acc, n_labeled, target_acc=0.85, cost_per_label=1.0):
    """
    Calculate cost savings of active learning.
    """
    active_arr = np.array(active_acc)
    random_arr = np.array(random_acc)
    n_arr = np.array(n_labeled)

    # Find labels needed for target accuracy
    active_idx = np.argmax(active_arr >= target_acc)
    random_idx = np.argmax(random_arr >= target_acc)

    if active_arr[active_idx] < target_acc:
        print(f"Active learning did not reach {target_acc:.1%}")
        return

    if random_arr[random_idx] < target_acc:
        print(f"Random sampling did not reach {target_acc:.1%}")
        return

    active_labels = n_arr[active_idx]
    random_labels = n_arr[random_idx]

    active_cost = active_labels * cost_per_label
    random_cost = random_labels * cost_per_label

    savings = random_cost - active_cost
    savings_pct = (savings / random_cost) * 100

    print(f"Target Accuracy: {target_acc:.1%}")
    print(f"\nActive Learning:")
    print(f"  Labels needed: {active_labels}")
    print(f"  Cost: ${active_cost:.2f}")
    print(f"\nRandom Sampling:")
    print(f"  Labels needed: {random_labels}")
    print(f"  Cost: ${random_cost:.2f}")
    print(f"\nSavings:")
    print(f"  Labels saved: {random_labels - active_labels}")
    print(f"  Cost saved: ${savings:.2f} ({savings_pct:.1f}%)")

cost_analysis(active_accuracies, random_accuracies, active_n_labeled,
              target_acc=0.85, cost_per_label=1.0)
```

Exercise 3.4: Multiple Runs (15 min)

Task: Run experiments with different random seeds

```python
def run_multiple_experiments(n_runs=5):
    """
    Run active learning multiple times with different seeds.
    """
    all_active = []
    all_random = []

    for run in range(n_runs):
        print(f"\nRun {run+1}/{n_runs}")

        # Generate new data split
        X, y = make_classification(
            n_samples=1000, n_features=20, n_classes=2,
            random_state=run
        )
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.3, random_state=run
        )

        # Run active learning
        X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
            X_train, y_train, n_initial=20
        )
        active_acc, _ = active_learning_loop(
            X_labeled.copy(), y_labeled.copy(), X_pool.copy(), y_pool.copy(),
            X_test, y_test, n_iterations=20, n_queries=10
        )

        # Run random sampling
        X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
            X_train, y_train, n_initial=20
        )
        random_acc, n_labels = random_sampling_loop(
            X_labeled, y_labeled, X_pool, y_pool,
            X_test, y_test, n_iterations=20, n_queries=10
        )

        all_active.append(active_acc)
        all_random.append(random_acc)

    # Calculate mean and std
    active_mean = np.mean(all_active, axis=0)
    active_std = np.std(all_active, axis=0)
    random_mean = np.mean(all_random, axis=0)
    random_std = np.std(all_random, axis=0)

    return active_mean, active_std, random_mean, random_std, n_labels

active_mean, active_std, random_mean, random_std, n_labels = run_multiple_experiments(n_runs=5)

print(f"\nFinal accuracy (mean ± std):")
print(f"Active: {active_mean[-1]:.3f} ± {active_std[-1]:.3f}")
print(f"Random: {random_mean[-1]:.3f} ± {random_std[-1]:.3f}")
```

Exercise 4.1: Using modAL Library (15 min)

Task: Implement active learning with modAL

```python
from modAL.models import ActiveLearner
from sklearn.ensemble import RandomForestClassifier

# Create initial split
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

# Create active learner
learner = ActiveLearner(
    estimator=RandomForestClassifier(random_state=42),
    X_training=X_labeled,
    y_training=y_labeled
)

# Active learning loop
accuracies = []
n_labeled = []

for iteration in range(20):
    # Evaluate
    accuracy = learner.score(X_test, y_test)
    accuracies.append(accuracy)
    n_labeled.append(len(learner.X_training))

    print(f"Iteration {iteration+1}: {len(learner.X_training)} labels, Accuracy = {accuracy:.3f}")

    if len(X_pool) == 0:
        break

    # Query
    n_query = min(10, len(X_pool))
    query_idx, query_instance = learner.query(X_pool, n_instances=n_query)

    # Teach
    learner.teach(X_pool[query_idx], y_pool[query_idx])

    # Remove from pool
    X_pool = np.delete(X_pool, query_idx, axis=0)
    y_pool = np.delete(y_pool, query_idx, axis=0)

print(f"\nFinal accuracy: {accuracies[-1]:.3f}")
```

Exercise 4.2: Custom Query Strategy with modAL (15 min)

Task: Define custom query strategy

```python
def entropy_query_strategy(classifier, X):
    """
    Custom query strategy using entropy.
    """
    probas = classifier.predict_proba(X)
    entropies = entropy(probas.T)
    return np.argsort(entropies)[-1]  # Return single index

# Create learner with custom strategy
learner_custom = ActiveLearner(
    estimator=RandomForestClassifier(random_state=42),
    query_strategy=entropy_query_strategy,
    X_training=X_labeled.copy(),
    y_training=y_labeled.copy()
)

# Run loop
X_pool_copy = X_pool.copy()
y_pool_copy = y_pool.copy()

for iteration in range(20):
    accuracy = learner_custom.score(X_test, y_test)
    print(f"Iteration {iteration+1}: Accuracy = {accuracy:.3f}")

    if len(X_pool_copy) == 0:
        break

    # Query single sample
    query_idx, _ = learner_custom.query(X_pool_copy)

    # Teach
    learner_custom.teach(
        X_pool_copy[query_idx].reshape(1, -1),
        y_pool_copy[query_idx].reshape(1,)
    )
```

Bonus Exercise 1: Query-by-Committee (20 min)

Task: Implement QBC from scratch

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

def query_by_committee(committee, X_pool, n_samples=10):
    """
    Query-by-Committee: select samples where models disagree.
    """
    # Get predictions from all models
    all_probas = []
    for model in committee:
        probas = model.predict_proba(X_pool)
        all_probas.append(probas)

    all_probas = np.array(all_probas)

    # Calculate average prediction
    avg_probas = all_probas.mean(axis=0)

    # Calculate vote entropy (disagreement)
    disagreements = entropy(avg_probas.T)

    # Select most disagreed samples
    query_idx = np.argsort(disagreements)[-n_samples:]

    return query_idx

# Run QBC
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=20
)

# Create committee
committee = [
    RandomForestClassifier(random_state=42),
    LogisticRegression(random_state=42, max_iter=1000),
    SVC(probability=True, random_state=42)
]

qbc_accuracies = []
n_labeled_qbc = []

for iteration in range(20):
    # Train all committee members
    for model in committee:
        model.fit(X_labeled, y_labeled)

    # Evaluate (use first model)
    accuracy = committee[0].score(X_test, y_test)
    qbc_accuracies.append(accuracy)
    n_labeled_qbc.append(len(X_labeled))

    print(f"Iteration {iteration+1}: Accuracy = {accuracy:.3f}")

    if len(X_pool) == 0:
        break

    # Query
    query_idx = query_by_committee(committee, X_pool, n_samples=10)

    # Update
    X_labeled = np.vstack([X_labeled, X_pool[query_idx]])
    y_labeled = np.hstack([y_labeled, y_pool[query_idx]])
    X_pool = np.delete(X_pool, query_idx, axis=0)
    y_pool = np.delete(y_pool, query_idx, axis=0)
```

Bonus Exercise 2: Real Dataset - MNIST (20 min)

Task: Apply active learning to MNIST

```python
from sklearn.datasets import load_digits

# Load digits dataset (8x8 images)
digits = load_digits()
X, y = digits.data, digits.target

# Use only 2 classes for binary classification
mask = (y == 3) | (y == 8)
X, y = X[mask], y[mask]
y = (y == 8).astype(int)  # 0 for digit 3, 1 for digit 8

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(f"Training samples: {len(X_train)}")
print(f"Classes: {np.unique(y_train)}")

# Run active learning
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=10
)

mnist_active_acc, mnist_n = active_learning_loop(
    X_labeled, y_labeled, X_pool, y_pool,
    X_test, y_test,
    n_iterations=15,
    n_queries=5
)

# Run random baseline
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(
    X_train, y_train, n_initial=10
)

mnist_random_acc, _ = random_sampling_loop(
    X_labeled, y_labeled, X_pool, y_pool,
    X_test, y_test,
    n_iterations=15,
    n_queries=5
)

# Plot
plot_learning_curves(mnist_active_acc, mnist_random_acc, mnist_n, title="MNIST Digits (3 vs 8)")
```

Bonus Exercise 3: Class Imbalance (20 min)

Task: Handle imbalanced datasets

```python
def class_balanced_uncertainty_sampling(model, X_pool, y_pool_predicted, n_samples=10):
    """
    Balance queries across classes.
    """
    probas = model.predict_proba(X_pool)
    uncertainties = 1 - np.max(probas, axis=1)

    # Get predicted classes
    pred_classes = np.argmax(probas, axis=1)

    selected = []
    n_classes = len(np.unique(pred_classes))
    samples_per_class = n_samples // n_classes

    for cls in np.unique(pred_classes):
        # Get samples predicted as this class
        cls_mask = pred_classes == cls
        cls_uncertainties = uncertainties[cls_mask]
        cls_indices = np.where(cls_mask)[0]

        if len(cls_indices) == 0:
            continue

        # Select most uncertain from this class
        top_k = min(samples_per_class, len(cls_indices))
        selected_idx = np.argsort(cls_uncertainties)[-top_k:]
        selected.extend(cls_indices[selected_idx])

    return np.array(selected)

# Create imbalanced dataset
X, y = make_classification(
    n_samples=1000,
    n_features=20,
    n_classes=2,
    weights=[0.9, 0.1],  # Imbalanced
    random_state=42
)

print(f"Class distribution: {np.bincount(y)}")

# Test balanced sampling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_labeled, y_labeled, X_pool, y_pool, _ = create_initial_split(X_train, y_train, n_initial=20)

model = LogisticRegression(random_state=42)
model.fit(X_labeled, y_labeled)

pred_pool = model.predict(X_pool)
balanced_idx = class_balanced_uncertainty_sampling(model, X_pool, pred_pool, n_samples=20)

print(f"Selected class distribution: {np.bincount(y_pool[balanced_idx])}")
```

Deliverables

Submit the following:

1. Python notebook with all exercises
2. Learning curve plots comparing:
   - Active learning vs random sampling
   - Different query strategies
   - Multiple runs with error bars
3. Analysis report including:
   - Cost-effectiveness analysis
   - Which strategy worked best and why
   - Characteristics of selected samples
4. Bonus: Results on MNIST or other real dataset

README.md should include:

Testing Checklist

Before submission:

- [ ] Uncertainty sampling implemented correctly
- [ ] Active learning loop runs without errors
- [ ] Random baseline implemented for comparison
- [ ] Learning curves plotted and saved
- [ ] Cost analysis completed
- [ ] Multiple runs show consistent improvement
- [ ] modAL library examples work
- [ ] Code is well-commented
- [ ] Results are reproducible (set random seeds)

Common Issues and Solutions

Issue: Active learning not improving over random

- Check if dataset is too easy
- Try different model (e.g., Random Forest)
- Increase query batch size
- Ensure initial split is small enough

Issue: Model accuracy not improving

- Check for bugs in oracle function
- Verify pool samples are being removed
- Ensure model is being retrained
- Check for label leakage

Issue: Slow execution

Resources

Libraries:

- modAL: https://modal-python.readthedocs.io/
- scikit-learn: https://scikit-learn.org/
- scipy: https://scipy.org/

Datasets:

- scikit-learn datasets: `sklearn.datasets`
- UCI ML Repository: https://archive.ics.uci.edu/ml/
- OpenML: https://www.openml.org/

Papers:

- Settles, "Active Learning Literature Survey" (2009)
- "A Survey of Deep Active Learning" (2020)

Excellent Work!

You've built a complete active learning system from scratch!

Next week: Data Augmentation

Questions? Office hours tomorrow 3-5 PM