style: @import "custom.css";

Monitoring & Continual Learning

Lab Session

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Lab Objectives

1. Detect data drift with Evidently
2. Calculate PSI manually
3. Monitor model predictions
4. Implement online learning with River
5. Build monitoring dashboard
6. Simulate and detect drift
7. Implement retraining pipeline

Setup

```
pip install evidently whylogs-python
pip install river scikit-learn pandas numpy
pip install plotly streamlit
pip install great-expectations
```

Exercise 1: Data Drift Detection with Evidently

Task: Detect drift between training and production data

Dataset: California Housing

Exercise 1: Load and Split Data

```python
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

# Load data
california = fetch_california_housing()
df = pd.DataFrame(california.data, columns=california.feature_names)
df['target'] = california.target

# Split: reference (train) and current (test + drift)
reference_data = df.sample(frac=0.6, random_state=42)
test_data = df.drop(reference_data.index)

# Create drifted data (simulate production drift)
drifted_data = test_data.copy()
# Add drift to MedInc feature
drifted_data['MedInc'] = drifted_data['MedInc'] * 1.3 + 0.5

print(f"Reference: {len(reference_data)}")
print(f"Current (drifted): {len(drifted_data)}")
```

Exercise 1: Generate Drift Report

```python
from evidently.report import Report
from evidently.metric_preset import DataDriftPreset

# Create drift report
drift_report = Report(metrics=[
    DataDriftPreset()
])

drift_report.run(
    reference_data=reference_data,
    current_data=drifted_data,
    column_mapping=None
)

# Save HTML report
drift_report.save_html('drift_report.html')

# Get results as dictionary
results = drift_report.as_dict()
print(f"Dataset drift detected: {results['metrics'][0]['result']['dataset_drift']}")
print(f"Number of drifted features: {results['metrics'][0]['result']['number_of_drifted_columns']}")
```

Exercise 1: Test Suite

```python
from evidently.test_suite import TestSuite
from evidently.test_preset import DataDriftTestPreset

# Create test suite
tests = TestSuite(tests=[
    DataDriftTestPreset()
])

tests.run(
    reference_data=reference_data,
    current_data=drifted_data
)

# Check results
test_results = tests.as_dict()
print(f"Tests passed: {test_results['summary']['by_status']['PASSED']}")
print(f"Tests failed: {test_results['summary']['by_status']['FAILED']}")

# Save HTML
tests.save_html('drift_tests.html')
```

Exercise 1: Feature-Level Analysis

```python
from evidently.metrics import ColumnDriftMetric

# Check each feature individually
for feature in reference_data.columns:
    if feature == 'target':
        continue

    single_feature_report = Report(metrics=[
        ColumnDriftMetric(column_name=feature)
    ])

    single_feature_report.run(
        reference_data=reference_data,
        current_data=drifted_data
    )

    result = single_feature_report.as_dict()
    drift_detected = result['metrics'][0]['result']['drift_detected']
    drift_score = result['metrics'][0]['result']['drift_score']

    if drift_detected:
        print(f"Drift in {feature}: score={drift_score:.3f}")
```

Exercise 2: Calculate PSI Manually

Task: Implement and use Population Stability Index

Exercise 2: PSI Implementation

```python
import numpy as np
import pandas as pd

def calculate_psi(expected, actual, bins=10):
    """
    Calculate PSI between two distributions

    Returns:
        psi: float, PSI value
        breakdown: DataFrame with per-bin PSI
    """
    # Handle edge cases
    if len(expected) == 0 or len(actual) == 0:
        return np.nan, None

    # Create bins based on expected distribution
    breakpoints = np.percentile(expected, np.linspace(0, 100, bins + 1))
    breakpoints = np.unique(breakpoints)  # Remove duplicates

    # Calculate percentages in each bin
    expected_percents = np.histogram(expected, bins=breakpoints)[0] / len(expected)
    actual_percents = np.histogram(actual, bins=breakpoints)[0] / len(actual)

    # Avoid division by zero
    expected_percents = np.where(expected_percents == 0, 0.0001, expected_percents)
    actual_percents = np.where(actual_percents == 0, 0.0001, actual_percents)

    # Calculate PSI for each bin
    psi_values = (actual_percents - expected_percents) * np.log(actual_percents / expected_percents)

    # Total PSI
    psi = np.sum(psi_values)

    # Create breakdown DataFrame
    breakdown = pd.DataFrame({
        'bin': range(len(psi_values)),
        'expected_pct': expected_percents,
        'actual_pct': actual_percents,
        'psi': psi_values
    })

    return psi, breakdown
```

Exercise 2: Test PSI on Features

```python
# Calculate PSI for each feature
psi_results = []

for feature in reference_data.columns:
    if feature == 'target':
        continue

    psi, breakdown = calculate_psi(
        reference_data[feature].values,
        drifted_data[feature].values,
        bins=10
    )

    psi_results.append({
        'feature': feature,
        'psi': psi
    })

# Create results DataFrame
psi_df = pd.DataFrame(psi_results).sort_values('psi', descending=True)
print(psi_df)

# Interpret results
print("\nDrift Assessment:")
for _, row in psi_df.iterrows():
    feature = row['feature']
    psi = row['psi']

    if psi < 0.1:
        status = "No drift"
    elif psi < 0.2:
        status = "Moderate drift"
    else:
        status = "Significant drift"

    print(f"{feature:15} PSI: {psi:.3f} - {status}")
```

Exercise 2: Visualize PSI

```python
import matplotlib.pyplot as plt

# PSI bar chart
fig, ax = plt.subplots(figsize=(10, 6))
bars = ax.barh(psi_df['feature'], psi_df['psi'])

# Color code by severity
colors = ['green' if psi < 0.1 else 'orange' if psi < 0.2 else 'red'
          for psi in psi_df['psi']]
for bar, color in zip(bars, colors):
    bar.set_color(color)

ax.axvline(x=0.1, color='orange', linestyle='--', label='Moderate threshold')
ax.axvline(x=0.2, color='red', linestyle='--', label='Significant threshold')
ax.set_xlabel('PSI')
ax.set_title('Feature Drift (PSI)')
ax.legend()
plt.tight_layout()
plt.savefig('psi_analysis.png')
plt.show()
```

Exercise 3: Prediction Monitoring

Task: Monitor model predictions over time

Exercise 3: Train Model and Generate Predictions

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Train model on reference data
model = RandomForestRegressor(n_estimators=100, random_state=42)
X_ref = reference_data.drop('target', axis=1)
y_ref = reference_data['target']
model.fit(X_ref, y_ref)

# Generate predictions on reference data
ref_predictions = model.predict(X_ref)

# Generate predictions on drifted data
X_drift = drifted_data.drop('target', axis=1)
y_drift = drifted_data['target']
drift_predictions = model.predict(X_drift)

# Evaluate
ref_r2 = r2_score(y_ref, ref_predictions)
drift_r2 = r2_score(y_drift, drift_predictions)

print(f"Reference R²: {ref_r2:.3f}")
print(f"Drifted R²: {drift_r2:.3f}")
print(f"Performance degradation: {(ref_r2 - drift_r2) / ref_r2 * 100:.1f}%")
```

Exercise 3: Monitor Prediction Distribution

```python
# Calculate PSI for predictions
pred_psi, pred_breakdown = calculate_psi(ref_predictions, drift_predictions)

print(f"Prediction PSI: {pred_psi:.3f}")

# Visualize prediction distributions
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

axes[0].hist(ref_predictions, bins=50, alpha=0.6, label='Reference', color='blue')
axes[0].hist(drift_predictions, bins=50, alpha=0.6, label='Drifted', color='red')
axes[0].set_xlabel('Predicted Value')
axes[0].set_ylabel('Frequency')
axes[0].legend()
axes[0].set_title('Prediction Distribution')

# Prediction error distribution
ref_errors = ref_predictions - y_ref
drift_errors = drift_predictions - y_drift

axes[1].hist(ref_errors, bins=50, alpha=0.6, label='Reference', color='blue')
axes[1].hist(drift_errors, bins=50, alpha=0.6, label='Drifted', color='red')
axes[1].set_xlabel('Prediction Error')
axes[1].set_ylabel('Frequency')
axes[1].legend()
axes[1].set_title('Error Distribution')

plt.tight_layout()
plt.show()
```

Exercise 4: Online Learning with River

Task: Implement online learning for streaming data

Dataset: Simulated data stream

Exercise 4: Create Data Stream

```python
from river import datasets
from river import linear_model
from river import metrics
from river import preprocessing

# Use Phishing dataset (online binary classification)
dataset = datasets.Phishing()

# Peek at first example
for x, y in dataset:
    print("Features:", x)
    print("Label:", y)
    break
```

Exercise 4: Online Learning

```
from river import compose

# Create online model pipeline
model = compose.Pipeline(
    ('scale', preprocessing.StandardScaler()),
    ('log_reg', linear_model.LogisticRegression())
)

# Track accuracy
metric = metrics.Accuracy()

# Learn from stream
predictions = []
actuals = []

for i, (x, y) in enumerate(dataset):
    # Predict before learning
    y_pred = model.predict_one(x)

    # Update metric
    if y_pred is not None:
        metric.update(y, y_pred)
        predictions.append(y_pred)
        actuals.append(y)

    # Learn from this example
    model.learn_one(x, y)

    # Print progress
    if (i + 1) % 250 == 0:
        print(f"Samples: {i+1}, Accuracy: {metric.get():.3f}")

print(f"\nFinal Accuracy: {metric.get():.3f}")
```

Exercise 4: Compare with Batch Learning

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import numpy as np

# Convert stream to arrays
X_list, y_list = [], []
dataset = datasets.Phishing()
for x, y in dataset:
    X_list.append(list(x.values()))
    y_list.append(y)

X = np.array(X_list)
y = np.array(y_list)

# Train batch model
batch_model = Pipeline([
    ('scaler', StandardScaler()),
    ('logistic', LogisticRegression())
])

# Use first 80% for training
split = int(len(X) * 0.8)
batch_model.fit(X[:split], y[:split])

# Test on remaining 20%
batch_score = batch_model.score(X[split:], y[split:])

print(f"Batch model accuracy: {batch_score:.3f}")
print(f"Online model accuracy: {metric.get():.3f}")
```

Exercise 5: Drift Detection with River

Task: Detect concept drift in a stream

Exercise 5: Stream with Concept Drift

```python
from river import drift
from river import synth

# Create synthetic stream with drift
stream = synth.ConceptDriftStream(
    stream=synth.Agrawal(classification_function=0, seed=42),
    drift_stream=synth.Agrawal(classification_function=2, seed=42),
    seed=42,
    position=5000,  # Drift occurs at position 5000
    width=500  # Drift transition width
)

# Initialize drift detector
drift_detector = drift.ADWIN()

# Model and metrics
model = linear_model.LogisticRegression()
metric = metrics.Accuracy()

# Track drift points
drift_points = []
accuracies = []

for i, (x, y) in enumerate(stream.take(10000)):
    # Predict
    y_pred = model.predict_one(x)

    # Check prediction correctness
    is_correct = (y == y_pred) if y_pred is not None else None

    # Update drift detector
    if is_correct is not None:
        in_drift = drift_detector.update(is_correct)

        if in_drift:
            print(f"Drift detected at sample {i}")
            drift_points.append(i)

    # Update metrics
    if y_pred is not None:
        metric.update(y, y_pred)

    accuracies.append(metric.get())

    # Learn
    model.learn_one(x, y)
```

Exercise 5: Visualize Drift Detection

```python
import matplotlib.pyplot as plt

# Plot accuracy over time
plt.figure(figsize=(12, 6))
plt.plot(accuracies, label='Accuracy', alpha=0.7)

# Mark drift points
for dp in drift_points:
    plt.axvline(x=dp, color='red', linestyle='--', alpha=0.5)

# Mark true drift location
plt.axvline(x=5000, color='green', linestyle='-', linewidth=2, label='True drift')

plt.xlabel('Samples')
plt.ylabel('Accuracy')
plt.title('Online Learning with Drift Detection')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

Exercise 6: Model Retraining Pipeline

Task: Implement automated retraining based on drift detection

Exercise 6: Retraining Logic

```python
class AutoRetrainingSystem:
    def __init__(self, model, psi_threshold=0.2):
        self.model = model
        self.psi_threshold = psi_threshold
        self.reference_data = None
        self.retrain_count = 0

    def set_reference(self, X, y):
        """Set reference data for drift detection"""
        self.reference_data = (X, y)
        self.model.fit(X, y)

    def predict_and_monitor(self, X_batch, y_batch=None):
        """Predict and check for drift"""
        # Make predictions
        predictions = self.model.predict(X_batch)

        # Check drift if we have reference data
        if self.reference_data is not None:
            drift_detected = self._check_drift(X_batch)

            if drift_detected:
                print(f"Drift detected! Retraining model...")
                self._retrain(X_batch, y_batch)

        return predictions

    def _check_drift(self, X_batch):
        """Check for data drift using PSI"""
        X_ref = self.reference_data[0]

        # Calculate PSI for each feature
        max_psi = 0
        for i in range(X_batch.shape[1]):
            psi, _ = calculate_psi(X_ref[:, i], X_batch[:, i])
            max_psi = max(max_psi, psi)

        return max_psi > self.psi_threshold

    def _retrain(self, X_new, y_new):
        """Retrain model with new data"""
        if y_new is not None:
            # Combine with reference data
            X_ref, y_ref = self.reference_data
            X_combined = np.vstack([X_ref, X_new])
            y_combined = np.concatenate([y_ref, y_new])

            # Retrain
            self.model.fit(X_combined, y_combined)

            # Update reference
            self.reference_data = (X_combined, y_combined)
            self.retrain_count += 1
```

Exercise 6: Test Retraining System

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

# Create initial dataset
X_init, y_init = make_classification(n_samples=1000, n_features=20, random_state=42)

# Initialize system
auto_system = AutoRetrainingSystem(
    model=RandomForestClassifier(n_estimators=100, random_state=42),
    psi_threshold=0.2
)

# Set reference data
auto_system.set_reference(X_init, y_init)

# Simulate production batches
for batch_id in range(10):
    # Generate batch (with increasing drift)
    drift_amount = batch_id * 0.1
    X_batch, y_batch = make_classification(
        n_samples=200,
        n_features=20,
        random_state=42 + batch_id,
        shift=drift_amount  # Simulate drift
    )

    # Predict and monitor
    predictions = auto_system.predict_and_monitor(X_batch, y_batch)

    print(f"Batch {batch_id}: {len(predictions)} predictions")

print(f"\nTotal retrains: {auto_system.retrain_count}")
```

Exercise 7: Monitoring Dashboard

Task: Build real-time monitoring dashboard with Streamlit

Exercise 7: Create Dashboard

```python
# Save as monitoring_dashboard.py
import streamlit as st
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from plotly.subplots import make_subplots

st.title("ML Model Monitoring Dashboard")

# Sidebar configuration
st.sidebar.header("Configuration")
psi_threshold = st.sidebar.slider("PSI Threshold", 0.0, 0.5, 0.2)
refresh_rate = st.sidebar.selectbox("Refresh Rate", [5, 10, 30, 60])

# Load monitoring data
@st.cache_data
def load_monitoring_data():
    # Simulate monitoring data
    dates = pd.date_range(start='2024-01-01', periods=30, freq='D')
    data = {
        'date': dates,
        'accuracy': 0.95 - np.random.rand(30) * 0.1,
        'latency_ms': 50 + np.random.rand(30) * 20,
        'requests': np.random.randint(1000, 5000, 30),
        'drift_score': np.random.rand(30) * 0.3
    }
    return pd.DataFrame(data)

monitoring_df = load_monitoring_data()
```

Exercise 7: Dashboard Metrics

```
# Key metrics
col1, col2, col3, col4 = st.columns(4)

with col1:
    current_acc = monitoring_df['accuracy'].iloc[-1]
    st.metric("Accuracy", f"{current_acc:.3f}",
              delta=f"{(current_acc - monitoring_df['accuracy'].iloc[-2]):.3f}")

with col2:
    current_latency = monitoring_df['latency_ms'].iloc[-1]
    st.metric("Latency (ms)", f"{current_latency:.1f}",
              delta=f"{(current_latency - monitoring_df['latency_ms'].iloc[-2]):.1f}")

with col3:
    total_requests = monitoring_df['requests'].sum()
    st.metric("Total Requests", f"{total_requests:,}")

with col4:
    current_drift = monitoring_df['drift_score'].iloc[-1]
    drift_status = "🟢 Normal" if current_drift < psi_threshold else "🔴 Alert"
    st.metric("Drift Status", drift_status, delta=f"{current_drift:.3f}")
```

Exercise 7: Performance Charts

```
# Performance over time
st.header("Performance Metrics")

fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=("Accuracy", "Latency", "Requests", "Drift Score")
)

fig.add_trace(
    go.Scatter(x=monitoring_df['date'], y=monitoring_df['accuracy'],
               mode='lines+markers', name='Accuracy'),
    row=1, col=1
)

fig.add_trace(
    go.Scatter(x=monitoring_df['date'], y=monitoring_df['latency_ms'],
               mode='lines+markers', name='Latency'),
    row=1, col=2
)

fig.add_trace(
    go.Bar(x=monitoring_df['date'], y=monitoring_df['requests'],
           name='Requests'),
    row=2, col=1
)

fig.add_trace(
    go.Scatter(x=monitoring_df['date'], y=monitoring_df['drift_score'],
               mode='lines+markers', name='Drift Score'),
    row=2, col=2
)

# Add threshold line for drift
fig.add_hline(y=psi_threshold, line_dash="dash", line_color="red",
              row=2, col=2)

fig.update_layout(height=600, showlegend=False)
st.plotly_chart(fig, use_container_width=True)
```

Exercise 7: Alerts Section

```
# Alerts
st.header("Active Alerts")

# Check for alerts
alerts = []

# Drift alert
if monitoring_df['drift_score'].iloc[-1] > psi_threshold:
    alerts.append({
        'severity': 'High',
        'type': 'Data Drift',
        'message': f"Drift score {monitoring_df['drift_score'].iloc[-1]:.3f} exceeds threshold {psi_threshold}"
    })

# Performance alert
if monitoring_df['accuracy'].iloc[-1] < 0.90:
    alerts.append({
        'severity': 'Medium',
        'type': 'Performance Degradation',
        'message': f"Accuracy dropped to {monitoring_df['accuracy'].iloc[-1]:.3f}"
    })

# Latency alert
if monitoring_df['latency_ms'].iloc[-1] > 70:
    alerts.append({
        'severity': 'Low',
        'type': 'High Latency',
        'message': f"Latency increased to {monitoring_df['latency_ms'].iloc[-1]:.1f}ms"
    })

# Display alerts
if alerts:
    for alert in alerts:
        if alert['severity'] == 'High':
            st.error(f"🔴 {alert['type']}: {alert['message']}")
        elif alert['severity'] == 'Medium':
            st.warning(f"🟡 {alert['type']}: {alert['message']}")
        else:
            st.info(f"🔵 {alert['type']}: {alert['message']}")
else:
    st.success("✅ No alerts - all systems normal")
```

Exercise 7: Feature Drift Details

```
# Feature-level drift
st.header("Feature Drift Analysis")

# Simulate feature drift scores
features = ['feature_1', 'feature_2', 'feature_3', 'feature_4', 'feature_5']
drift_scores = np.random.rand(len(features)) * 0.4

feature_drift_df = pd.DataFrame({
    'Feature': features,
    'Drift Score': drift_scores,
    'Status': ['🔴 Alert' if d > psi_threshold else '🟢 Normal' for d in drift_scores]
})

st.dataframe(feature_drift_df, use_container_width=True)

# Run dashboard with: streamlit run monitoring_dashboard.py
```

Challenge 1: Complete Monitoring System

Task: Build end-to-end monitoring system

Requirements:

1. Detect data drift with Evidently
2. Calculate PSI for all features
3. Monitor prediction distribution
4. Track performance metrics
5. Generate daily reports
6. Send email alerts on drift

Challenge 2: Adaptive Retraining

Task: Implement intelligent retraining strategy

Requirements:

1. Detect drift with multiple methods
2. Decide when to retrain (cost vs benefit)
3. Use active learning to label samples
4. Retrain incrementally
5. A/B test new model
6. Track model versions

Challenge 3: Real-Time Drift Detection

Task: Implement streaming drift detection

Requirements:

1. Use River for online drift detection
2. Handle concept drift
3. Adapt model in real-time
4. Maintain performance metrics
5. Visualize drift points
6. Compare with batch detection

Best Practices

- Monitor continuously, not just during deployment
- Set appropriate thresholds based on business impact
- Automate alerts and retraining
- Version all models and datasets
- Track both data and model metrics
- Test retraining pipeline regularly
- Document incident response procedures
- Balance automation with human oversight

Common Issues

False Positives:

- Too sensitive thresholds
- Natural variation flagged as drift
- Solution: Tune thresholds, use multiple metrics

Missed Drift:

- Thresholds too high
- Gradual drift not detected
- Solution: Multiple detection methods

Retraining Too Frequently:

- Expensive

Key Takeaways

1. Evidently makes drift detection easy
2. PSI is simple but effective
3. Monitor both data and predictions
4. Online learning enables real-time adaptation
5. Automation is critical for production
6. Dashboard provides visibility
7. Have clear incident response plan

Additional Resources

Tools:

- Evidently: evidentlyai.com/docs
- River: riverml.xyz/latest
- WhyLogs: whylogs.readthedocs.io
- Streamlit: streamlit.io

Tutorials:

- Evidently Tutorial: github.com/evidentlyai/evidently
- River Tutorials: riverml.xyz/latest/examples
- ML Monitoring Guide: mlinproduction.com

Next Steps

Production deployment requires:

- Robust monitoring
- Automated retraining
- Incident response
- Model versioning
- Performance tracking

You now have the complete ML pipeline!

- Data collection → Validation → Training
- Deployment → Monitoring → Retraining

Congratulations on completing the course!