style: @import "custom.css";

section { justify-content: flex-start; }

Monitoring & Continual Learning

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Today's Agenda

- Model monitoring in production
- Data drift detection
- Concept drift detection
- Model performance monitoring
- Alerting and incident response
- Continual learning strategies
- Online learning
- Tools: Evidently, WhyLogs, River

Why Monitor ML Models?

Models degrade over time:

- Data distributions change
- User behavior evolves
- External world shifts
- Software/hardware changes

Without monitoring:

- Silent failures
- Degraded user experience
- Business impact

ML Monitoring vs Traditional Monitoring

Traditional Monitoring:

- CPU, memory, latency
- Error rates, uptime
- Infrastructure health

ML Monitoring:

- Prediction distribution
- Feature distribution
- Model performance metrics
- Data quality
- Inference patterns

What to Monitor

1. **System Metrics**:
    - Latency (p50, p95, p99)
    - Throughput (requests/sec)
    - Resource usage
    - Error rates
2. **Data Metrics**:
    - Feature distributions
    - Missing values
    - Outliers
    - Data drift

Data Drift

Definition: Input data distribution changes over time

$$P_{train}(X) \neq P_{prod}(X)$$

Examples:

- Camera quality improves → image features change
- User demographics shift → feature distributions change
- Sensor calibration drifts → measurement distributions change

Impact: Model performance degrades

Types of Data Drift

Covariate Shift:

- Features change: $P(X)$ changes
- Relationship stable: $P(Y|X)$ stable
- Example: new camera model

Concept Drift:

- Relationship changes: $P(Y|X)$ changes
- Features stable: $P(X)$ stable
- Example: fraud patterns evolve

Label Drift:

- Target changes: $P(Y)$ changes

Detecting Data Drift

Statistical Tests:

- Kolmogorov-Smirnov test
- Chi-square test
- Population Stability Index (PSI)

Distance Metrics:

- Kullback-Leibler divergence
- Wasserstein distance
- Jensen-Shannon divergence

Tools:

- Evidently

Population Stability Index (PSI)

```python
import numpy as np

def calculate_psi(expected, actual, bins=10):
    """
    Calculate PSI between two distributions
    PSI < 0.1: No significant change
    0.1 < PSI < 0.2: Moderate change
    PSI > 0.2: Significant change
    """
    # Create bins
    breakpoints = np.percentile(expected, np.linspace(0, 100, bins+1))

    # 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
    psi = np.sum((actual_percents - expected_percents) *
                 np.log(actual_percents / expected_percents))

    return psi
```

Using PSI

```
# Training data feature
train_feature = np.random.normal(100, 15, 10000)

# Production data (drifted)
prod_feature = np.random.normal(105, 18, 5000)

psi = calculate_psi(train_feature, prod_feature)
print(f"PSI: {psi:.3f}")

if psi < 0.1:
    print("No significant drift")
elif psi < 0.2:
    print("Moderate drift - monitor closely")
else:
    print("Significant drift - retrain model")
```

Evidently for Drift Detection

```python
from evidently.report import Report
from evidently.metric_preset import DataDriftPreset
import pandas as pd

# Reference data (training)
reference = pd.read_csv('train_data.csv')

# Current data (production)
current = pd.read_csv('prod_data.csv')

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

report.run(reference_data=reference, current_data=current)

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

# Get drift scores
drift_results = report.as_dict()
print(f"Dataset drift: {drift_results['metrics'][0]['result']['dataset_drift']}")
```

Evidently 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, current_data=current)

# Check if tests passed
print(tests.as_dict()['summary'])

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

WhyLogs for Data Logging

```python
import whylogs as why
from whylogs.core import DatasetProfileView

# Log reference data
reference_profile = why.log(reference).profile().view()

# Log production data
current_profile = why.log(current).profile().view()

# Calculate drift
from whylogs.core.metrics.metrics import MetricConfig
from whylogs.core.relations import Relationship

# Compare profiles
drift_report = reference_profile.diff(current_profile)

# Get summary
summary = drift_report.to_pandas()
print(summary)
```

Feature-Level Drift Detection

```python
from scipy.stats import ks_2samp

def detect_drift_per_feature(reference_df, current_df, threshold=0.05):
    drift_features = []

    for column in reference_df.columns:
        if reference_df[column].dtype in ['int64', 'float64']:
            # Kolmogorov-Smirnov test
            statistic, p_value = ks_2samp(
                reference_df[column],
                current_df[column]
            )

            if p_value < threshold:
                drift_features.append({
                    'feature': column,
                    'p_value': p_value,
                    'statistic': statistic
                })

    return pd.DataFrame(drift_features)

drift_df = detect_drift_per_feature(reference, current)
print(drift_df.sort_values('p_value'))
```

Concept Drift Detection

Challenge: Need ground truth labels

- Labels arrive delayed
- Labels expensive to obtain

Strategies:

1. Monitor proxy metrics (CTR, engagement)
2. Use delayed feedback when available
3. Statistical tests on predictions
4. Ensemble disagreement

Prediction Drift Monitoring

```python
# Monitor prediction distribution
import matplotlib.pyplot as plt

def monitor_predictions(train_preds, prod_preds):
    fig, axes = plt.subplots(1, 2, figsize=(12, 4))

    # Plot distributions
    axes[0].hist(train_preds, bins=50, alpha=0.5, label='Training')
    axes[0].hist(prod_preds, bins=50, alpha=0.5, label='Production')
    axes[0].set_xlabel('Prediction')
    axes[0].set_ylabel('Frequency')
    axes[0].legend()
    axes[0].set_title('Prediction Distribution')

    # Calculate drift
    psi = calculate_psi(train_preds, prod_preds)
    axes[1].text(0.5, 0.5, f'PSI: {psi:.3f}',
                 ha='center', va='center', fontsize=20)
    axes[1].set_title('Drift Score')
    axes[1].axis('off')

    plt.tight_layout()
    return fig, psi
```

Model Performance Monitoring

With labels (batch/delayed):

```python
from sklearn.metrics import accuracy_score, f1_score

def monitor_performance(y_true, y_pred, date):
    metrics = {
        'date': date,
        'accuracy': accuracy_score(y_true, y_pred),
        'f1_score': f1_score(y_true, y_pred, average='weighted')
    }
    return metrics

# Log daily
daily_metrics = []
for date, (y_true, y_pred) in production_data.groupby('date'):
    metrics = monitor_performance(y_true, y_pred, date)
    daily_metrics.append(metrics)

metrics_df = pd.DataFrame(daily_metrics)
```

Confidence Monitoring

When labels unavailable:

```python
def monitor_confidence(predictions_with_proba):
    confidences = np.max(predictions_with_proba, axis=1)

    metrics = {
        'mean_confidence': np.mean(confidences),
        'median_confidence': np.median(confidences),
        'low_confidence_rate': np.mean(confidences < 0.6),
        'high_confidence_rate': np.mean(confidences > 0.9)
    }

    return metrics

# Monitor over time
for date, batch in production_batches.items():
    conf_metrics = monitor_confidence(batch['predictions'])
    print(f"{date}: Mean confidence: {conf_metrics['mean_confidence']:.3f}")

    if conf_metrics['low_confidence_rate'] > 0.3:
```

Alerting System

```python
class ModelMonitor:
    def __init__(self, psi_threshold=0.2, conf_threshold=0.6):
        self.psi_threshold = psi_threshold
        self.conf_threshold = conf_threshold
        self.alerts = []

    def check_drift(self, reference, current):
        psi = calculate_psi(reference, current)
        if psi > self.psi_threshold:
            alert = {
                'type': 'data_drift',
                'severity': 'high' if psi > 0.3 else 'medium',
                'psi': psi,
                'message': f'Significant drift detected (PSI: {psi:.3f})'
            }
            self.alerts.append(alert)
            self.send_alert(alert)

    def check_confidence(self, predictions):
        mean_conf = np.mean(np.max(predictions, axis=1))
        if mean_conf < self.conf_threshold:
            alert = {
                'type': 'low_confidence',
                'severity': 'medium',
                'confidence': mean_conf,
                'message': f'Low confidence detected: {mean_conf:.3f}'
            }
            self.alerts.append(alert)
            self.send_alert(alert)

    def send_alert(self, alert):
        # Integration with Slack, PagerDuty, email, etc.
        print(f"ALERT: {alert['message']}")
```

Continual Learning

Goal: Update model as new data arrives

Benefits:

- Adapt to distribution shifts
- Improve over time
- Reduce staleness

Challenges:

- Catastrophic forgetting
- Label availability
- Computational cost
- Model versioning

Continual Learning Strategies

1. **Periodic Retraining**:
    - Retrain on all data (old + new)
    - Simple, effective
    - Computationally expensive
2. **Incremental Learning**:
    - Update model with new data only
    - Faster, less resources
    - Risk of catastrophic forgetting
3. **Online Learning**:
    - Update after each example

Periodic Retraining

```python
import schedule
import time

def retrain_model():
    # Load all historical data
    train_data = load_all_data()

    # Train model
    model = RandomForestClassifier()
    model.fit(train_data['X'], train_data['y'])

    # Evaluate on recent data
    recent_data = load_recent_data()
    score = model.score(recent_data['X'], recent_data['y'])
    print(f"Retrained model score: {score:.3f}")

    # Save new version
    save_model(model, version=get_next_version())

    # Deploy
    deploy_model(model)

# Schedule weekly retraining
schedule.every().sunday.at("02:00").do(retrain_model)

while True:
    schedule.run_pending()
    time.sleep(3600)  # Check every hour
```

Incremental Learning with scikit-learn

```python
from sklearn.linear_model import SGDClassifier

# Initialize model
model = SGDClassifier(loss='log_loss', random_state=42)

# Initial training
model.partial_fit(X_initial, y_initial, classes=np.unique(y_initial))

# Incremental updates
for X_batch, y_batch in new_data_batches:
    model.partial_fit(X_batch, y_batch)

    # Evaluate periodically
    score = model.score(X_val, y_val)
    print(f"Validation score: {score:.3f}")
```

Online Learning with River

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

# Create online model
model = (
    preprocessing.StandardScaler() |
    linear_model.LogisticRegression()
)

# Track performance
metric = metrics.Accuracy()

# Learn from stream
for x, y in stream:
    # Predict
    y_pred = model.predict_one(x)

    # Update metric
    metric.update(y, y_pred)

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

    # Log performance
    if len(metric) % 1000 == 0:
        print(f"Samples: {len(metric)}, Accuracy: {metric.get():.3f}")
```

Handling Concept Drift with River

```python
from river import drift

# Initialize drift detector
drift_detector = drift.ADWIN()

# Online learning with drift detection
model = linear_model.LogisticRegression()
metric = metrics.Accuracy()

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

    # Detect drift
    in_drift = drift_detector.update(y == y_pred)

    if in_drift:
        print(f"Drift detected at sample {len(metric)}")
        # Reset or retrain model
        model = linear_model.LogisticRegression()

    # Learn
    model.learn_one(x, y)
    metric.update(y, y_pred)
```

Replay Buffers

Problem: Catastrophic forgetting

Solution: Maintain buffer of past examples

```python
from collections import deque
import random

class ReplayBuffer:
    def __init__(self, max_size=10000):
        self.buffer = deque(maxlen=max_size)

    def add(self, x, y):
        self.buffer.append((x, y))

    def sample(self, n=32):
        return random.sample(self.buffer, min(n, len(self.buffer)))

# Usage
buffer = ReplayBuffer(max_size=10000)

for x_new, y_new in new_data:
    # Add to buffer
    buffer.add(x_new, y_new)
```

Active Learning for Labeling

```python
from modAL.models import ActiveLearner

# Initialize with small labeled set
learner = ActiveLearner(
    estimator=RandomForestClassifier(),
    X_training=X_initial,
    y_training=y_initial
)

# Query most uncertain examples
for X_unlabeled_batch in production_data:
    # Find uncertain examples
    query_idx, _ = learner.query(X_unlabeled_batch, n_instances=10)

    # Get labels (human annotation)
    X_to_label = X_unlabeled_batch[query_idx]
    y_labels = get_labels(X_to_label)  # Human labeling

    # Update model
    learner.teach(X_to_label, y_labels)

    # Evaluate
    score = learner.score(X_val, y_val)
    print(f"Model performance: {score:.3f}")
```

Model Versioning

```python
import mlflow

def train_and_version_model(X, y, version_name):
    with mlflow.start_run():
        # Train model
        model = RandomForestClassifier(n_estimators=100)
        model.fit(X, y)

        # Log parameters
        mlflow.log_params({
            'n_estimators': 100,
            'data_size': len(X),
            'version': version_name
        })

        # Log metrics
        score = model.score(X_val, y_val)
        mlflow.log_metric('val_accuracy', score)

        # Log model
        mlflow.sklearn.log_model(
            model,
            'model',
            registered_model_name=f'wine_classifier_{version_name}'
        )

    return model
```

A/B Testing New Models

```
import random

class ABTestingPredictor:
    def __init__(self, model_a, model_b, traffic_split=0.5):
        self.model_a = model_a
        self.model_b = model_b
        self.traffic_split = traffic_split
        self.results_a = []
        self.results_b = []

    def predict(self, X):
        # Route traffic
        if random.random() < self.traffic_split:
            model = self.model_a
            group = 'A'
        else:
            model = self.model_b
            group = 'B'

        prediction = model.predict(X)

        # Log for analysis
        self.log_prediction(X, prediction, group)

        return prediction

    def get_results(self):
        return {
            'A': self.results_a,
            'B': self.results_b
        }
```

Shadow Deployment

```python
class ShadowPredictor:
    def __init__(self, production_model, shadow_model):
        self.production_model = production_model
        self.shadow_model = shadow_model
        self.comparisons = []

    def predict(self, X):
        # Production prediction (served to user)
        prod_pred = self.production_model.predict(X)

        # Shadow prediction (logged only)
        shadow_pred = self.shadow_model.predict(X)

        # Compare predictions
        self.comparisons.append({
            'production': prod_pred,
            'shadow': shadow_pred,
            'agreement': prod_pred == shadow_pred
        })

        return prod_pred

    def analyze_shadow(self):
        agreement_rate = np.mean([c['agreement'] for c in self.comparisons])
        print(f"Agreement rate: {agreement_rate:.3f}")
```

Monitoring Dashboard

```python
import streamlit as st
import plotly.graph_objects as go

def create_monitoring_dashboard():
    st.title("Model Monitoring Dashboard")

    # Load metrics
    metrics_df = load_metrics()

    # Performance over time
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=metrics_df['date'],
        y=metrics_df['accuracy'],
        name='Accuracy'
    ))
    st.plotly_chart(fig)

    # Drift detection
    st.header("Data Drift")
    drift_df = load_drift_metrics()
    st.dataframe(drift_df)

    # Alerts
    st.header("Active Alerts")
    alerts = load_alerts()
    for alert in alerts:
        st.error(f"{alert['type']}: {alert['message']}")
```

Complete Monitoring Pipeline

```python
class ProductionMonitor:
    def __init__(self, model, reference_data):
        self.model = model
        self.reference_data = reference_data
        self.predictions = []
        self.features = []

    def predict_and_log(self, X):
        # Make prediction
        y_pred = self.model.predict(X)

        # Log features and predictions
        self.features.append(X)
        self.predictions.append(y_pred)

        return y_pred

    def run_daily_checks(self):
        # Combine recent data
        recent_features = np.vstack(self.features[-1000:])
        recent_preds = np.concatenate(self.predictions[-1000:])

        # Check data drift
        psi_scores = {}
        for i, col in enumerate(self.reference_data.columns):
            psi = calculate_psi(
                self.reference_data[col],
                recent_features[:, i]
            )
            psi_scores[col] = psi

        # Check prediction drift
        pred_psi = calculate_psi(
            self.reference_data['predictions'],
            recent_preds
        )

        # Generate report
        self.generate_report(psi_scores, pred_psi)
```

Incident Response

When drift detected:

1. **Investigate**:
    - Which features drifted?
    - When did drift start?
    - What caused it?
2. **Assess Impact**:
    - Performance degradation?
    - User complaints?
    - Business metrics affected?
3. **Mitigate**:

Data Quality Monitoring

```python
from great_expectations.dataset import PandasDataset

def validate_production_data(df):
    ge_df = PandasDataset(df)

    # Check for missing values
    ge_df.expect_column_values_to_not_be_null('important_feature')

    # Check value ranges
    ge_df.expect_column_values_to_be_between('age', 0, 120)

    # Check distributions
    ge_df.expect_column_mean_to_be_between('income', 30000, 100000)

    # Get validation results
    results = ge_df.get_expectation_suite()

    # Alert on failures
    if not results['success']:
        send_alert("Data quality issues detected")

    return results
```

Best Practices

1. **Monitor continuously**: Don't wait for complaints
2. **Set thresholds**: Know when to act
3. **Automate alerts**: Real-time notification
4. **Version everything**: Models, data, code
5. **A/B test changes**: Validate improvements
6. **Document incidents**: Learn from failures
7. **Retrain regularly**: Stay current
8. **Track lineage**: Data → Model → Predictions

Monitoring Checklist

- [ ] System metrics (latency, throughput)
- [ ] Data drift detection
- [ ] Prediction drift monitoring
- [ ] Performance metrics (when available)
- [ ] Data quality checks
- [ ] Alerting system
- [ ] Incident response plan
- [ ] Regular retraining schedule
- [ ] Model versioning
- [ ] Monitoring dashboard

Common Pitfalls

Over-reacting to drift:

- Small fluctuations normal
- Set appropriate thresholds

Under-monitoring:

- Silent failures
- Gradual degradation

Forgetting system metrics:

- ML metrics not enough
- Infrastructure matters

No incident response:

Tools Comparison

Evidently:

- Great for reports
- Easy to use
- Good visualizations

WhyLogs:

- Efficient logging
- Scalable
- Integration with MLflow

River:

- Online learning

Key Takeaways

- Models degrade over time - monitor actively
- Data drift: distribution changes
- Concept drift: relationship changes
- Use statistical tests (PSI, KS-test)
- Continual learning: periodic vs incremental vs online
- Tools: Evidently, WhyLogs, River
- Version models and track lineage
- Automate monitoring and alerting
- Have incident response plan

Resources

Tools:

- Evidently: evidentlyai.com
- WhyLogs: whylogs.readthedocs.io
- River: riverml.xyz
- Great Expectations: greatexpectations.io

Papers:

- "A Survey on Concept Drift Adaptation"
- "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift"

Courses:

- Stanford CS 329S: ML Systems Design

Lab Preview

Hands-on exercises:

1. Detect data drift with Evidently
2. Monitor predictions with WhyLogs
3. Implement PSI calculation
4. Online learning with River
5. Build monitoring dashboard
6. Simulate drift and detect
7. Implement alerting system