Subsampling and Data Drift in MoDeVa

July 5, 2025

Outline

What is Subsampling?

- **Definition:** Technique to create a smaller dataset that represents the original data
- **Applications:**
  - Reducing computational requirements
  - Balancing class distributions
  - Creating training/validation splits
  - Accelerating development and experimentation
- **Key considerations:**
  - Maintaining representative distributions
  - Preserving important patterns and relationships
  - Accounting for class imbalance

What is Data Drift?

- **Definition:** Changes in the statistical properties of data over time or between samples
- **Types of drift:**
  - **Feature drift:** Changes in input distributions
  - **Label drift:** Changes in target variable distributions
  - **Concept drift:** Changes in relationships between inputs and outputs
- **Importance:**
  - Affects model performance and validity
  - Critical for monitoring production systems
  - Essential for assessing subsampling quality

Subsampling and Data Drift in MoDeVa

The `DataSet` class in Modeva provides two key functions:

- `DataSet.subsample_random`
  - Performs random subsampling with options
  - Supports stratification to maintain class distributions
  - Returns indices for further manipulation
- `DataSet.data_drift_test`
  - Assesses distributional differences between datasets
  - Supports multiple statistical distance metrics
  - Provides quantitative measures of distribution change

Random Subsampling Overview

`DataSet.subsample_random` provides:

- **Simple random sampling:** Each data point has equal probability of selection
- **Stratified sampling:** Maintains proportions of specified categorical variables
- **Flexible sample sizes:** Specify absolute count or proportion of original data
- **Shuffling options:** Control randomization behavior

Basic Usage

```
# Simple random subsampling
subsampler = dataset.subsample_random(
    dataset="main", sample_size=1000,  # number of samples
    shuffle=True, random_state=42  # For reproducibility
)
idx = subsampler.value["sample_idx"] # Get subsampled indices
# Apply subsample to the dataset
dataset.set_active_samples(dataset="main", sample_idx=idx)
```

Stratified Subsampling

- **Definition:** Sampling that preserves the proportions of specified variables
- **Benefits:**
  - Maintains class distributions in imbalanced datasets
  - Ensures representation of all important subgroups
  - Reduces sampling bias

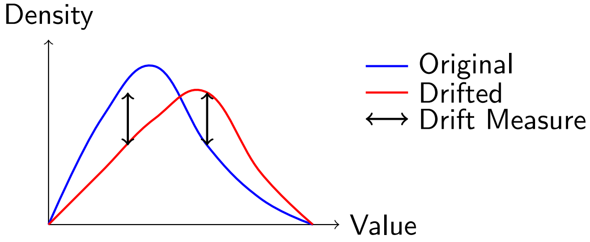
Stratified Sampling

```
# Stratified subsampling by a categorical variable
subsampler = dataset.subsample_random(
    dataset="main", sample_size=0.3,  # 30% of original data
    stratify="Gender",  # Stratify by Gender column
    shuffle=True, random_state=42
)
idx = subsampler.value["sample_idx"] # Get subsampled indices
# Apply subsample to the dataset
dataset.set_active_samples(dataset="main", sample_idx=idx)
```

Distribution Drift Overview

`DataSet.data_drift_test` assesses distributional differences using:

- **Population Stability Index (PSI):** Based on Kullback-Leibler divergence
- **Wasserstein Distance 1D (WD1):** Based on cumulative distribution differences
- **Kolmogorov-Smirnov (KS) Distance:** Based on maximum distribution difference



Population Stability Index (PSI)

- **Definition:** A measure based on the Kullback-Leibler divergence
- **Calculation:**
  1. Bin both datasets (original and comparison)
  2. Calculate proportions in each bin (p_i and q_i)
  3. Apply formula:

$$\mathrm{PSI} = \sum_{i=1}^{B} (p_i - q_i) \ln \left(\frac{p_i}{q_i} \right)$$

- **Interpretation:**
  - $\mathrm{PSI} < 0.1$: No significant change
  - $0.1 \leq \mathrm{PSI} < 0.2$: Moderate change
  - $\mathrm{PSI} \geq 0.2$: Significant change
- **Note:** PSI is sensitive to the binning method (equal width or equal quantile)

Wasserstein Distance 1D (WD1)

- **Definition:** The 1-dimensional Earth Mover's Distance
- **Calculation:** Integral of absolute difference between CDFs

$$\mathrm{WD1} = \int_{-\infty}^{\infty} |F(x) - G(x)| dx$$

  where $F(x)$ and $G(x)$ are the cumulative distribution functions
- **Advantages:**
  - No binning required
  - Accounts for the "distance" between values
  - More intuitive for continuous variables
- **Implementation:** Uses `scipy.stats.wasserstein_distance`

Kolmogorov-Smirnov (KS) Distance

- **Definition:** Maximum absolute difference between CDFs
- **Calculation:**

$$\mathrm{KS} = \max_x |F(x) - G(x)|$$

  where $F(x)$ and $G(x)$ are the cumulative distribution functions
- **Advantages:**
  - No binning required
  - Simple interpretation
  - Sensitive to differences in any part of the distribution
- **Implementation:** Uses `scipy.stats.ks_2samp`

Implementing Data Drift Tests in Modeva

Basic Usage

```
# Test for data drift between original and subsampled data
drift_results = dataset.data_drift_test(
    features=["Age", "Income", "Education"],  # Features to test
    base_dataset="main",  # Original dataset
    target_dataset="main",  # Dataset to compare with
    base_sample_idx=None,  # Use all samples in base
    target_sample_idx=idx,  # Use subsampled indices
    distance_metrics=["psi", "ks", "wd1"]  # Metrics to calculate
)
# Examine results for each feature
for feature in drift_results.value:
    print(f"Feature: {feature}")
    for metric, value in drift_results.value[feature].items():
        print(f"  {metric}: {value}")
```

Use Case: Training-Test Split Validation

```
# Split dataset into training (70%) and test (30%) sets
train_sampler = dataset.subsample_random(
    dataset="main", sample_size=0.7, stratify="Target",  # Stratify by target
    shuffle=True, random_state=42
)
# Get training indices
train_idx = train_sampler.value["sample_idx"]
# Create test indices (all samples not in training)
all_indices = set(range(len(dataset.data["main"])))
test_idx = list(all_indices - set(train_idx))
# Test for distribution drift between train and test sets
drift_results = dataset.data_drift_test(
    features=["Feature1", "Feature2", "Feature3"], base_dataset="main",
    target_dataset="main", base_sample_idx=train_idx,
    target_sample_idx=test_idx, distance_metrics=["psi", "ks"])
```

Use Case: Model Monitoring

```
production_dataset = DataSet(production_data)
# Test for data drift between training and production data
drift_results = dataset.data_drift_test(
    features=all_model_features,  # All features used in the model
    base_dataset="main",  # Original training data
    target_dataset=production_dataset.dataset_id,  # New data
    distance_metrics=["psi", "ks", "wd1"]
)
# Threshold-based alerting
drift_detected = False
for feature, metrics in drift_results.value.items():
    if metrics["psi"] > 0.2 or metrics["ks"] > 0.1:
        print(f"Significant drift detected in {feature}!")
        drift_detected = True
```

Use Case: Imbalanced Data Handling

```
# Check class distribution
class_counts = dataset.data["main"]["Target"].value_counts()
print(f"Original class distribution: {class_counts}")

# Create a balanced dataset through stratified downsampling
# First, find the minority class count
min_class_count = min(class_counts)
# Create subsampled balanced dataset
balanced_sampler = dataset.subsample_random(
    dataset="main", sample_size=min_class_count * len(class_counts), # Total 
    stratify="Target", shuffle=True, random_state=42
)
balanced_idx = balanced_sampler.value["sample_idx"]
dataset.set_active_samples(dataset="main", sample_idx=balanced_idx)
```

Best Practices for Subsampling

1. **Stratify when important:** Use stratification for target variables and key features
2. **Set appropriate sample size:**
   - Large enough to capture important patterns
   - Small enough to gain computational benefits
3. **Validate distribution preservation:** Use data drift metrics to verify representativeness
4. **Use consistent random seeds:** For reproducibility
5. **Consider multiple subsamples:** For sensitivity analysis
6. **Document subsampling methodology:** Track parameters and decisions

Best Practices for Data Drift Assessment

1. **Choose appropriate metrics:**
   - PSI: Good for categorical variables and binned features
   - KS: Sensitive to any distributional differences
   - WD1: Better for continuous variables, accounts for value differences
2. **Establish meaningful thresholds:** Define acceptable drift levels
3. **Focus on important features:** Prioritize model inputs and key business variables
4. **Consider feature relationships:** Individual feature drift may not capture joint distributions
5. **Combine with model performance metrics:** Connect drift to impact

Complete Workflow Example

```
# 1. Create a stratified subsample (e.g., for faster experimentation)
subsampler = dataset.subsample_random(
    dataset="main", sample_size=5000,
    stratify="Target", shuffle=True, random_state=42
)
subsample_idx = subsampler.value["sample_idx"]

# 2. Verify distribution preservation
drift_results = dataset.data_drift_test(
    features=important_features, base_dataset="main",
    target_dataset="main",
    base_sample_idx=None,  # All samples
    target_sample_idx=subsample_idx,
    distance_metrics=["psi", "ks", "wd1"]
)
```

Complete Workflow Example

```
# 3. Create a report on distribution drift
drift_df = pd.DataFrame([
    {
        "Feature": feature,
        "PSI": metrics["psi"],
        "KS": metrics["ks"],
        "WD1": metrics["wd1"]
    }
    for feature, metrics in drift_results.value.items()
])
# 4. Apply subsample if drift is acceptable
if drift_df["PSI"].max() < 0.1 and drift_df["KS"].max() < 0.1:
    dataset.set_active_samples(dataset="main", sample_idx=subsample_idx)
    print("Subsample applied - distribution is well-preserved")
else:
    print("Warning: Significant drift detected in subsample")
```

Summary

- **Subsampling** enables efficient data analysis while preserving important distributions
- **Data drift assessment** quantifies distributional differences between datasets
- **Modeva provides:**
  - `DataSet.subsample_random`: Flexible random sampling with stratification
  - `DataSet.data_drift_test`: Multiple metrics for drift quantification
- **Key metrics:**
  - PSI: Based on binned Kullback-Leibler divergence
  - WD1: Based on integrated CDF differences
  - KS: Based on maximum CDF difference
- **Applications:** Training-test splits, model monitoring, imbalanced data handling