Data Collection and Labeling

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra

Table of contents

1 Module Overview

1.1 Four Core Components

1. **Data Collection** - Tools and techniques for gathering data from diverse sources
2. **Data Validation** - Ensuring data quality and reliability
3. **Data Labeling** - Annotating datasets with ground truth
4. **Data Augmentation** - Expanding datasets strategically

Key Insight

Quality data is the foundation of successful AI systems. Garbage in, garbage out!

2 Part 1: Data Collection

Instrumenting and Logging Data Sources

2.1 Why Data Collection Matters

- **Real-world AI systems** depend on continuous data flow
- **User behavior** changes over time → models need fresh data
- **Production systems** require automated collection pipelines
- **Debugging** often requires understanding what data was seen

2.2 Common Data Sources

2.2.1 Digital Sources

- Web applications
- Mobile apps
- IoT devices
- APIs and databases

2.2.2 Physical Sources

- Sensors
- Cameras
- Microphones
- Manual entry

2.3 Instrumentation: The Foundation

Instrumentation = Adding code to collect data about system behavior

2.3.1 Key Principles

1. **Minimal Performance Impact** - Don't slow down production systems
2. **Comprehensive Coverage** - Capture all relevant events
3. **Privacy-Aware** - Respect user privacy and regulations (GDPR, CCPA)
4. **Structured Logging** - Consistent formats for easy parsing

2.4 Basic Instrumentation Example

```python
import logging
import json
from datetime import datetime

def log_user_action(user_id, action, metadata):
    event = {
        "timestamp": datetime.utcnow().isoformat(),
        "user_id": user_id,
        "action": action,
        "metadata": metadata
    }
    logging.info(json.dumps(event))
```

2.5 Web Analytics Tools

2.5.1 Google Analytics 4

```
// Track custom events
gtag('event', 'search', {
  'search_term': query,
  'results_count': results.length
});
```

Pros: Free, Rich ecosystem, Real-time dashboards **Cons:** Privacy concerns, Limited customization

2.5.2 Mixpanel

```
// Track with properties
mixpanel.track('Video Played', {
  'video_id': video.id,
  'duration': video.length,
  'quality': '1080p'
});
```

Pros: Detailed analytics, Cohort analysis **Cons:** Costly at scale

2.6 Self-Hosted Analytics

2.6.1 Why Self-Host?

- **Data ownership** - Complete control over your data
- **Privacy compliance** - GDPR-friendly by design
- **No tracking cookies** - Lightweight and fast
- **Cost-effective** - Predictable infrastructure costs

2.7 Plausible & Umami Examples

```
// Plausible: Simple, privacy-focused
<script defer data-domain="yourdomain.com"
        src="https://plausible.io/js/script.js"></script>

plausible('signup', {props: {plan: 'premium'}});
```

```
// Umami: Open-source alternative
<script async src="https://analytics.yourdomain.com/umami.js"
        data-website-id="your-website-id"></script>

umami.track('button-click', {button: 'subscribe'});
```

2.8 Application Performance Monitoring

2.8.1 Sentry

```
import sentry_sdk

sentry_sdk.init(
    dsn="your-dsn",
    traces_sample_rate=1.0
)

try:
    process_data()
except Exception as e:
    sentry_sdk.capture_exception(e)
```

2.8.2 Datadog

```
from datadog import initialize, statsd

initialize(
    api_key='your-key',
    app_key='your-app-key'
)

statsd.increment('api.requests')
statsd.histogram('api.latency', 245)
```

2.9 Database Event Logging

```sql
-- PostgreSQL: Track all changes to users table
CREATE TABLE user_audit (
    audit_id SERIAL PRIMARY KEY,
    user_id INT,
    action VARCHAR(10),
    old_data JSONB,
    new_data JSONB,
    changed_at TIMESTAMP DEFAULT NOW()
);

CREATE OR REPLACE FUNCTION audit_user_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'UPDATE') THEN
        INSERT INTO user_audit(user_id, action, old_data, new_data)
        VALUES (NEW.id, 'UPDATE', row_to_json(OLD), row_to_json(NEW));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

2.10 Change Data Capture (CDC)

CDC = Capturing and streaming database changes in real-time

2.10.1 Debezium Example

```json
{
  "name": "inventory-connector",
  "config": {
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "database.hostname": "postgres",
    "database.port": "5432",
    "table.include.list": "public.orders,public.customers",
    "topic.prefix": "dbserver1"
  }
}
```

Benefits: Real-time streaming, No app changes, Complete history

2.11 OpenTelemetry: Industry Standard

```python
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider

trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer(__name__)

@tracer.start_as_current_span("process_request")
def process_request(user_id):
    span = trace.get_current_span()
    span.set_attribute("user.id", user_id)
    # Your logic here
```

2.12 Web Scraping with Scrapy

```python
import scrapy

class ProductSpider(scrapy.Spider):
    name = 'products'
    start_urls = ['https://example.com/products']

    custom_settings = {
        'DOWNLOAD_DELAY': 2,  # Respectful scraping
        'CONCURRENT_REQUESTS': 1
    }

    def parse(self, response):
        for product in response.css('div.product'):
            yield {
                'name': product.css('h2::text').get(),
                'price': product.css('span.price::text').get(),
                'rating': product.css('div.rating::attr(data-rating)').get()
            }
```

2.13 Ethical Web Scraping

1. **Check robots.txt** - Respect website's scraping policy
2. **Rate Limiting** - Don't overwhelm servers
3. **User-Agent** - Identify yourself honestly
4. **Terms of Service** - Read and comply with ToS
5. **Personal Data** - Respect privacy laws (GDPR, CCPA)

⚠ Warning

Scraping can violate ToS or copyright. Always consult legal counsel for commercial use.

2.14 Streaming Data with Kafka

```
from kafka import KafkaProducer, KafkaConsumer
import json

# Producer: Collect data
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

event = {
    'user_id': '12345',
    'event_type': 'page_view',
    'page': '/products'
}
producer.send('user-events', event)

# Consumer: Process data
consumer = KafkaConsumer('user-events')
```

```python
for message in consumer:
    process_event(message.value)
```

2.15 Data Collection Best Practices

2.15.1 Schema Design with Pydantic

```python
from pydantic import BaseModel, Field
from datetime import datetime

class UserEvent(BaseModel):
    user_id: str
    event_type: str
    timestamp: datetime = Field(default_factory=datetime.utcnow)
    properties: dict
    session_id: Optional[str] = None
```

2.16 Sampling Strategies

```python
import random

class SamplingCollector:
    def __init__(self, sample_rate=0.1):
        self.sample_rate = sample_rate

    def should_collect(self, event):
        if random.random() < self.sample_rate:
            return True
        if event.get('priority') == 'high':
            return True
        return False
```

When to use: High-traffic systems where full collection is expensive

2.17 Privacy and Compliance

```python
import hashlib

class PrivacyAwareCollector:
    def __init__(self, pii_fields=['email', 'phone']):
        self.pii_fields = pii_fields

    def anonymize(self, data):
        anonymized = data.copy()
        for field in self.pii_fields:
            if field in anonymized:
                value = str(anonymized[field])
                hashed = hashlib.sha256(value.encode()).hexdigest()
                anonymized[field] = hashed
        return anonymized
```

3 Part 2: Data Validation

Ensuring Data Quality and Reliability

3.1 Why Data Validation Matters

3.1.1 The Cost of Bad Data

- **Garbage In, Garbage Out**
- **Silent Failures**
- **Expensive Debugging**
- **Lost Trust**

3.1.2 Real-World Impact

E-commerce: Missing IDs → 30% drop in CTR **Medical:** Wrong units → Misdiagnoses **Fraud Detection:** Duplicates → 45% false positives

3.2 Types of Data Quality Issues

1. **Completeness** - Missing or null values
2. **Accuracy** - Incorrect or outdated values
3. **Consistency** - Contradictory data

4. **Timeliness** - Stale or outdated data
5. **Uniqueness** - Duplicate records
6. **Validity** - Violates business rules

3.3 Data Validation with Pydantic

```python
from pydantic import BaseModel, Field, validator, EmailStr

class UserEvent(BaseModel):
    user_id: str = Field(..., min_length=1, max_length=100)
    email: EmailStr
    age: int = Field(..., ge=0, le=150)
    event_type: str

    @validator('event_type')
    def validate_event_type(cls, v):
        allowed = ['click', 'view', 'purchase', 'signup']
        if v not in allowed:
            raise ValueError(f'event_type must be one of {allowed}')
        return v

    @validator('timestamp')
    def timestamp_not_future(cls, v):
        if v > datetime.utcnow():
            raise ValueError('timestamp cannot be in the future')
        return v
```

3.4 Using Pydantic

```python
try:
    event = UserEvent(
        user_id="user_123",
        email="user@example.com",
        age=25,
        event_type="click",
        timestamp=datetime.utcnow()
    )
    print("  Valid event")
```

```python
except ValueError as e:
    print("  Validation errors:", e.json())
```

3.5 Great Expectations Framework

```python
import great_expectations as gx
from great_expectations.dataset import PandasDataset

df = pd.read_csv('user_events.csv')
dataset = PandasDataset(df)

# Define expectations
dataset.expect_column_values_to_not_be_null('user_id')
dataset.expect_column_values_to_be_unique('event_id')
dataset.expect_column_values_to_be_in_set(
    'event_type',
    ['click', 'view', 'purchase']
)
dataset.expect_column_values_to_be_between('age', 0, 150)

# Validate
results = dataset.validate()
```

3.6 Pandera: Statistical Validation

```python
import pandera as pa
from pandera import Column, Check

schema = pa.DataFrameSchema({
    "user_id": Column(str, checks=[
        Check.str_length(min_value=1, max_value=100)
    ]),
    "age": Column(int, checks=[
        Check.in_range(min_value=0, max_value=150),
        Check(lambda s: s.mean() > 18)
    ]),
    "purchase_amount": Column(float, checks=[
        Check.greater_than_or_equal_to(0)
    ], nullable=True)
})

validated_df = schema.validate(df)
```

3.7 Custom Validation Pipelines

```
from dataclasses import dataclass
from typing import List

@dataclass
class ValidationResult:
    passed: bool
    errors: List[str]
    warnings: List[str]

class DataValidationPipeline:
    def __init__(self):
        self.validators = []

    def add_validator(self, validator):
        self.validators.append(validator)
        return self

    def validate(self, df):
        all_errors = []
        for validator in self.validators:
            result = validator.validate(df)
            all_errors.extend(result.errors)
        return ValidationResult(
            passed=len(all_errors) == 0,
            errors=all_errors,
            warnings=[]
        )
```

3.8 Statistical Distribution Validation

```
class DistributionValidator:
    def __init__(self, reference_df, columns):
        self.reference_stats = {
            col: {
                'mean': reference_df[col].mean(),
                'std': reference_df[col].std()
            }
            for col in columns
        }

    def validate(self, df):
        errors = []
        for col, ref_stats in self.reference_stats.items():
            current_mean = df[col].mean()
            mean_shift = abs(current_mean - ref_stats['mean']) / ref_stats['std']

            if mean_shift > 2:
                errors.append(
                    f"{col}: Mean shifted by {mean_shift:.2f} std deviations"
                )
        return ValidationResult(passed=len(errors) == 0, errors=errors, warnings=[])
```

3.9 Data Quality Monitoring

```
import sentry_sdk
from datadog import statsd

class DataQualityMonitor:
    def monitor(self, df, dataset_name):
        result = self.pipeline.validate(df)

        # Send metrics
        statsd.gauge(f'{dataset_name}.row_count', len(df))
        statsd.gauge(f'{dataset_name}.validation.errors', len(result.errors))

        # Alert on failures
        if not result.passed:
            sentry_sdk.capture_message(
                f"Data validation failed for {dataset_name}",
                extras={"errors": result.errors}
            )
```

4 Part 3: Data Labeling

Annotating Datasets with Ground Truth

4.1 What is Data Labeling?

Data Labeling: Adding meaningful tags or annotations to raw data

4.1.1 Why It's Critical

- **Supervised Learning** requires labeled examples
- **Quality labels** → Better models
- **Consistent labels** → Reliable evaluation
- **Cost:** Often 60-80% of ML project time and budget

4.2 Types of Labeling Tasks

4.2.1 Classification

- Image: "cat" vs "dog"
- Text: "spam" vs "not spam"
- Audio: speaker identification

4.2.2 Object Detection

- Bounding boxes
- Keypoint annotation
- Polygon segmentation

4.2.3 Sequence Labeling

- Named Entity Recognition
- Part-of-speech tagging
- Time series anomalies

4.2.4 Structured Prediction

- Semantic segmentation
- Dependency parsing
- Relationship extraction

4.3 Label Studio

Label Studio: Open-source, multi-modal annotation platform

4.3.1 Key Features

- Multi-modal: Images, text, audio, video, time series
- Custom interfaces with XML config
- ML-assisted labeling and active learning
- Collaboration features
- Export: JSON, CSV, COCO, Pascal VOC, YOLO

4.3.2 Installation

```
pip install label-studio
label-studio start
```

4.4 Label Studio: Image Classification

```
<View>
  <Image name="image" value="$image_url"/>
  <Choices name="choice" toName="image">
    <Choice value="Cat"/>
    <Choice value="Dog"/>
    <Choice value="Other"/>
  </Choices>
</View>
```

4.5 Label Studio: Object Detection

```
<View>
  <Image name="image" value="$image_url"
         zoom="true" zoomControl="true"/>
  <RectangleLabels name="label" toName="image">
    <Label value="Person" background="red"/>
    <Label value="Car" background="blue"/>
    <Label value="Bicycle" background="green"/>
  </RectangleLabels>
</View>
```

4.6 Label Studio: NER

```
<View>
  <Text name="text" value="$text"/>
  <Labels name="label" toName="text">
    <Label value="Person" background="red"/>
    <Label value="Organization" background="blue"/>
    <Label value="Location" background="green"/>
    <Label value="Date" background="orange"/>
  </Labels>
</View>
```

Example: "Apple Inc. CEO Tim Cook announced the new iPhone in Cupertino"

4.7 Alternative Labeling Tools

4.7.1 Labelbox

- Enterprise features
- Model-assisted labeling
- **Cons:** Expensive, vendor lock-in

4.7.2 CVAT

- Open-source by Intel
- Video annotation
- **Cons:** Setup complexity

4.7.3 Prodigy

- By spaCy creators
- Active learning built-in
- **Cons:** License cost

4.7.4 Scale AI / SageMaker

- Managed services
- Human workforce included
- **Cons:** Very expensive

4.8 Inter-Annotator Agreement

Problem: Different annotators may label differently

Solution: Measure agreement to ensure quality

4.8.1 Cohen's Kappa ( )

Measures agreement between **two annotators** accounting for chance

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Interpretation:

- 0.0-0.20: Slight
- 0.21-0.40: Fair
- 0.41-0.60: Moderate
- 0.61-0.80: Substantial
- 0.81-1.00: Almost perfect

4.9 Cohen's Kappa Implementation

```python
from sklearn.metrics import cohen_kappa_score

annotator1 = ['spam', 'ham', 'spam', 'ham', ...]
annotator2 = ['spam', 'ham', 'ham', 'ham', ...]

kappa = cohen_kappa_score(annotator1, annotator2)
print(f"Cohen's Kappa: {kappa:.3f}")
```

Example Output:

```
Cohen's Kappa: 0.745
→ Substantial agreement
```

4.10 Fleiss' Kappa: Multiple Annotators

```python
from statsmodels.stats.inter_rater import fleiss_kappa
import numpy as np

# Matrix: rows = items, columns = category counts
data = np.array([
    [3, 0],  # Item 1: all 3 annotators said "ham"
    [2, 1],  # Item 2: 2 said "ham", 1 said "spam"
    [0, 3],  # Item 3: all 3 said "spam"
])

kappa = fleiss_kappa(data, method='fleiss')
print(f"Fleiss' Kappa: {kappa:.3f}")
```

4.11 Improving Agreement

4.11.1 1. Clear Guidelines

- Define each label precisely
- Provide examples and counter-examples
- Document edge cases
- Create decision trees for ambiguous cases

4.11.2 2. Training Phase

- Use gold standard datasets
- Review disagreements together
- Iterative calibration sessions

4.11.3 3. Regular Consensus Meetings

- Discuss disputed items
- Update guidelines based on patterns
- Track improvement over time

4.12 Sampling Strategies

4.12.1 Random Sampling

```python
import random
random.sample(dataset, n=100)
```

Pros: Unbiased **Cons:** May miss rare classes

4.12.2 Stratified Sampling

```python
from sklearn.model_selection import train_test_split
_, sampled, _, _ = train_test_split(
    dataset, labels,
    train_size=len(dataset)-100,
    stratify=labels
)
```

Pros: Ensures class representation

4.13 Snowball Sampling

```python
from sklearn.neighbors import NearestNeighbors

class SnowballSampler:
    def sample(self, labeled_data, unlabeled_data, n=100):
        # Vectorize
        labeled_vectors = self.vectorizer.fit_transform(labeled_data)
        unlabeled_vectors = self.vectorizer.transform(unlabeled_data)

        # Find nearest neighbors
        nn = NearestNeighbors(n_neighbors=5)
        nn.fit(unlabeled_vectors)
        distances, indices = nn.kneighbors(labeled_vectors)

        return [unlabeled_data[i] for i in indices.flatten()[:n]]
```

Use case: Finding more examples of a rare class

4.14 Active Learning

Active Learning: Intelligently select which examples to label next

4.14.1 Process

1. Train model on small labeled set
2. Find **most informative** unlabeled examples
3. Label those examples
4. Retrain model
5. Repeat

Goal: Achieve good performance with minimal labeling

4.15 Uncertainty Sampling

```python
class UncertaintySampler:
    def sample(self, X_unlabeled, n=10):
        probs = self.model.predict_proba(X_unlabeled)

        # Higher entropy = more uncertain
        uncertainties = -np.sum(probs * np.log(probs + 1e-10), axis=1)
        # Select top-n most uncertain
        uncertain_indices = np.argsort(uncertainties)[-n:]
        return uncertain_indices
```

4.16 Active Learning Loop

```
def active_learning_loop(X_labeled, y_labeled, X_unlabeled, budget=100):
    model = MultinomialNB()
    sampler = UncertaintySampler(model)

    for iteration in range(budget // 10):
        # Train model
        model.fit(X_labeled, y_labeled)

        # Select informative examples
        indices = sampler.sample(X_unlabeled, n=10)

        # Get human labels
        new_X = [X_unlabeled[i] for i in indices]
        new_y = get_human_labels(new_X)

        # Update datasets
        X_labeled.extend(new_X)
        y_labeled.extend(new_y)

        # Evaluate
        accuracy = evaluate_model(model)
        print(f"Iteration {iteration}: Accuracy = {accuracy:.3f}")
```

4.17 Active Learning Benefits

4.17.1 Cost Reduction

Random Sampling: 10,000 labels → 85% accuracy

Active Learning: 2,000 labels → 85% accuracy

Savings: 80% fewer labels!

4.17.2 Real-world Savings

- 10k labels × \$0.50 = \$5,000
- 2k labels × \$0.50 = \$1,000
- **Saved: \$4,000**

Key Insight: Active learning reaches target accuracy with 5-10× fewer labels

4.18 Weak Supervision with Snorkel

Weak Supervision: Use noisy/heuristic labels instead of manual annotation

```
from snorkel.labeling import labeling_function

SPAM = 1
HAM = 0
ABSTAIN = -1

@labeling_function()
def lf_contains_money(x):
    money_keywords = ['$', 'money', 'cash', 'prize']
    return SPAM if any(kw in x.text.lower() for kw in money_keywords) else ABSTAIN

@labeling_function()
def lf_short_message(x):
    return SPAM if len(x.text.split()) < 10 else ABSTAIN

@labeling_function()
def lf_known_sender(x):
    trusted_domains = ['company.com', 'university.edu']
    return HAM if any(d in x.sender for d in trusted_domains) else ABSTAIN
```

4.19 Combining Weak Labels

```
from snorkel.labeling.model import LabelModel

# Apply labeling functions
lfs = [lf_contains_money, lf_short_message, lf_known_sender]
L_train = applier.apply(df=df)
```

```python
# Combine noisy labels
label_model = LabelModel(cardinality=2)
label_model.fit(L_train=L_train, n_epochs=500)

# Get predictions
predicted_labels = label_model.predict(L=L_train)
```

Advantages: Fast, Flexible, No manual labeling needed

4.20 Pre-labeling with Models

```python
class PreLabelingPipeline:
    def pre_label(self, unlabeled_data, confidence_threshold=0.9):
        predictions = self.model.predict_proba(unlabeled_data)

        auto_labeled = []
        needs_review = []

        for i, probs in enumerate(predictions):
            max_prob = max(probs)

            if max_prob >= confidence_threshold:
                auto_labeled.append({
                    'data': unlabeled_data[i],
                    'label': np.argmax(probs),
                    'source': 'model'
                })
            else:
                needs_review.append(unlabeled_data[i])

        return auto_labeled, needs_review
```

5 Part 4: Data Augmentation

Expanding Training Datasets Strategically

5.1 Why Data Augmentation?

5.1.1 The Problem

- Limited labeled data is expensive
- Class imbalance leads to biased models
- Overfitting on small datasets
- Rare events underrepresented

5.1.2 The Solution

Data Augmentation: Creating new training examples by applying transformations

Benefits: Increase dataset size, Improve generalization, Reduce overfitting, Balance classes

5.2 Image Data Augmentation

```python
from torchvision import transforms

transform = transforms.Compose([
    # Geometric
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.RandomResizedCrop(size=224, scale=(0.8, 1.0)),

    # Color
    transforms.ColorJitter(
        brightness=0.2, contrast=0.2,
        saturation=0.2, hue=0.1
    ),

    # Blurring
    transforms.GaussianBlur(kernel_size=3),

    # Normalization
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
])
```

5.3 Advanced: Albumentations

```python
import albumentations as A

transform = A.Compose([
    A.RandomResizedCrop(height=224, width=224),
    A.HorizontalFlip(p=0.5),
    A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.1, rotate_limit=15),
    A.ElasticTransform(p=0.3),

    # Color
    A.RandomBrightnessContrast(p=0.5),
    A.HueSaturationValue(p=0.5),

    # Blur and noise
    A.OneOf([
        A.MotionBlur(p=0.5),
        A.GaussianBlur(p=0.5),
    ], p=0.3),
    A.GaussNoise(p=0.3),

    # Weather
    A.RandomRain(p=0.2),
    A.RandomFog(p=0.2),
])
```

5.4 Cutout, Mixup, CutMix

5.4.1 Cutout

Random rectangular masks

```python
A.CoarseDropout(
    max_holes=8,
    max_height=32,
    p=0.5
)
```

5.4.2 Mixup

Blend two images

```
def mixup(x1, y1, x2, y2):
    lam = np.random.beta(0.2, 0.2)
    x = lam * x1 + (1 - lam) * x2
    y = lam * y1 + (1 - lam) * y2
    return x, y
```

5.4.3 CutMix

Paste regions

```
def cutmix(x1, y1, x2, y2):
    # Cut and paste
    # region from x2 to x1
    ...
```

5.5 Text Data Augmentation

5.5.1 Synonym Replacement

```
from nltk.corpus import wordnet

def get_synonyms(word):
    synonyms = set()
    for syn in wordnet.synsets(word):
        for lemma in syn.lemmas():
            synonyms.add(lemma.name())
    return list(synonyms)

def synonym_replacement(text, n=2):
    words = text.split()
    random_indices = random.sample(range(len(words)), n)

    for idx in random_indices:
        synonyms = get_synonyms(words[idx])
        if synonyms:
            words[idx] = random.choice(synonyms)
    return ' '.join(words)
```

5.6 EDA: Easy Data Augmentation

```
class EDA:
    @staticmethod
    def random_insertion(words, n=1):
        """Insert n random synonyms"""
        ...

    @staticmethod
    def random_swap(words, n=1):
        """Swap n pairs of words"""
        ...

    @staticmethod
    def random_deletion(words, p=0.1):
        """Delete each word with probability p"""
        return [w for w in words if random.random() > p]
```

5.7 Back-Translation

```
from transformers import MarianMTModel, MarianTokenizer

class BackTranslator:
    def __init__(self, intermediate_lang='fr'):
        # English → French
        self.forward_model = MarianMTModel.from_pretrained(
            f'Helsinki-NLP/opus-mt-en-{intermediate_lang}'
        )
        # French → English
        self.backward_model = MarianMTModel.from_pretrained(
            f'Helsinki-NLP/opus-mt-{intermediate_lang}-en'
        )

    def augment(self, text):
        # Translate to French and back
        intermediate = self.forward_model.translate(text)
        return self.backward_model.translate(intermediate)
```

Result: “The weather is beautiful” → “The weather is nice”

5.8 Contextual Augmentation with BERT

```
from transformers import pipeline

class ContextualAugmentor:
    def __init__(self):
        self.unmasker = pipeline('fill-mask', model='bert-base-uncased')

    def augment(self, text):
        words = text.split()
        mask_idx = random.randint(0, len(words) - 1)

        masked_words = words.copy()
        masked_words[mask_idx] = '[MASK]'

        predictions = self.unmasker(' '.join(masked_words))

        return [pred['sequence'] for pred in predictions[:3]]
```

5.9 Audio Augmentation

```
import librosa

class AudioAugmentor:
    @staticmethod
    def add_noise(audio, noise_factor=0.005):
        noise = np.random.randn(len(audio))
        return audio + noise_factor * noise

    @staticmethod
    def time_stretch(audio, rate=1.0):
        return librosa.effects.time_stretch(audio, rate=rate)
    @staticmethod
    def pitch_shift(audio, sr=22050, n_steps=2):
        return librosa.effects.pitch_shift(audio, sr=sr, n_steps=n_steps)
```

5.10 Time Series Augmentation

```
class TimeSeriesAugmentor:
    @staticmethod
    def jittering(x, sigma=0.03):
        """Add Gaussian noise"""
        return x + np.random.normal(0, sigma, x.shape)

    @staticmethod
    def scaling(x, sigma=0.1):
        """Multiply by random factor"""
        factor = np.random.normal(1, sigma, size=(x.shape[0], 1))
        return x * factor

    @staticmethod
    def time_warping(x, sigma=0.2):
        """Warp time dimension"""
        # Use cubic spline interpolation
        ...
```

5.11 SMOTE for Tabular Data

SMOTE: Synthetic Minority Over-sampling Technique

```
from imblearn.over_sampling import SMOTE

# Imbalanced dataset: {0: 900, 1: 100}
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Now balanced: {0: 900, 1: 900}
```

How it works:

1. For each minority sample, find k nearest neighbors

2. Randomly select one neighbor
3. Create synthetic sample along line segment

5.12 SMOTE Variants

5.12.1 ADASYN

Adaptive Synthetic Sampling

```python
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(random_state=42)
X_res, y_res = adasyn.fit_resample(X, y)
```

Advantage: Generates more samples in harder-to-learn regions

5.12.2 BorderlineSMOTE

Focus on decision boundary

```python
from imblearn.over_sampling import BorderlineSMOTE

bsmote = BorderlineSMOTE(random_state=42)
X_res, y_res = bsmote.fit_resample(X, y)
```

Advantage: More conservative, focuses on boundary

5.13 Generative Models

```python
from diffusers import StableDiffusionPipeline

pipe = StableDiffusionPipeline.from_pretrained(
    "stabilityai/stable-diffusion-2-1"
)

def generate_augmented_images(prompt, n=10):
    images = []
    for i in range(n):
        image = pipe(prompt, num_inference_steps=50).images[0]
        images.append(image)
    return images

# Generate synthetic training data
synthetic = generate_augmented_images(
    "A high-quality photo of a defective product",
    n=100
)
```

Use cases: Rare defects, medical imaging, artistic styles

5.14 Best Practices

5.14.1 1. Domain-Appropriate

Good: Horizontal flip for general images **Bad:** Horizontal flip for text/digits (changes meaning)

5.14.2 2. Preserve Semantics

Bad: Extreme rotation for digits ($6 \rightarrow 9$) **Good:** Slight rotation ($\pm 15°$)

5.14.3 3. Validation Split First

```
#   Correct
train, val = train_test_split(data)
aug_train = augment(train)  # Only augment training

#   Wrong (data leakage!)
aug_data = augment(data)
train, val = train_test_split(aug_data)
```

5.15 Monitor Augmentation Impact

```python
def evaluate_augmentation(model, train_data, val_data, aug_levels):
    results = []

    for strength in aug_levels:
        aug_train = augment(train_data, strength=strength)
        model.fit(aug_train)
        val_acc = model.evaluate(val_data)
        results.append({'strength': strength, 'accuracy': val_acc})

    # Plot results to find optimal augmentation
    plt.plot([r['strength'] for r in results],
             [r['accuracy'] for r in results])
    plt.xlabel('Augmentation Strength')
    plt.ylabel('Validation Accuracy')
```

5.16 Class-Specific Augmentation

```python
class ClassBalancedAugmentor:
    def __init__(self, target_samples_per_class=1000):
        self.target = target_samples_per_class

    def augment(self, X, y):
        X_aug, y_aug = [], []

        for cls in np.unique(y):
            X_cls = X[y == cls]
            current_count = len(X_cls)

            if current_count < self.target:
                n_to_generate = self.target - current_count
                augmented = self.generate_samples(X_cls, n_to_generate)
                X_aug.extend(augmented)
                y_aug.extend([cls] * len(augmented))

        return np.array(X_aug), np.array(y_aug)
```

6 Summary

6.1 Key Takeaways

6.1.1 Data Collection

- Instrument systems for automatic capture
- Use appropriate tools (analytics, APM, CDC)
- Implement buffering, batching, error handling
- Respect privacy and comply with regulations

6.1.2 Data Validation

- Validate early and often
- Use schema validation (Pydantic, Pandera)
- Monitor data quality metrics
- Set up alerts for anomalies

6.2 Key Takeaways (cont.)

6.2.1 Data Labeling

- Use appropriate tools (Label Studio, CVAT)
- Measure inter-annotator agreement (Cohen's Kappa)
- Apply smart sampling (active learning)
- Leverage weak supervision when possible

6.2.2 Data Augmentation

- Choose domain-appropriate transformations
- Preserve label semantics
- Augment only training data
- Monitor impact on performance

❗ Important

High-quality data is the foundation of successful AI systems. Invest time in getting it right!

6.3 Hands-On Lab

6.3.1 Complete Data Pipeline

Tasks:

1. Data Collection (30 min) - Scrape reviews, set up logging
2. Data Validation (30 min) - Create schemas, build validation pipeline
3. Data Labeling (45 min) - Label Studio, calculate agreement, active learning
4. Data Augmentation (30 min) - Apply EDA, back-translation, measure impact

Deliverable: Jupyter notebook with complete pipeline

6.4 Additional Resources

6.4.1 Documentation

- Label Studio: https://labelstud.io/guide/
- Great Expectations: https://docs.greatexpectations.io/
- Albumentations: https://albumentations.ai/docs/
- Snorkel: https://www.snorkel.org/

6.4.2 Key Papers

- "SMOTE: Synthetic Minority Over-sampling Technique" (2002)
- "Snorkel: Rapid Training Data Creation with Weak Supervision" (2017)
- "Active Learning Literature Survey" (2009)

7 Questions?

7.1 Thank You!

Next: Reproducibility & Versioning

Contact: nipun.batra@iitgn.ac.in