# Feature Engineering for Battery OCV Analysis

## Practical Applications on A123 Battery Low Current Open Circuit Voltage Data

---

## Introduction

This document demonstrates comprehensive feature engineering techniques applied to A123 battery low current Open Circuit Voltage (OCV) data collected at different temperatures (-10°C to 50°C). We'll apply the fundamental concepts from machine learning feature engineering to transform raw battery testing data into meaningful features for analysis and modeling.

The dataset contains approximately 30,000-32,000 data points per temperature, with 18-19 features including voltage, current, capacity, energy, and temperature measurements across multiple test cycles.

---

## 1. Categorical Features

### 1.1 Temperature as Categorical Variable

The most obvious categorical feature in our dataset is temperature, which represents discrete test conditions rather than continuous thermal measurements.

**Raw Data Representation:**

```python
temperatures = [-10, 0, 10, 20, 25, 30, 40, 50]  # Celsius
```

**Problem with Direct Numerical Encoding:** Using direct numerical mapping (e.g., -10°C = 1, 0°C = 2, etc.) would incorrectly imply:

- Mathematical relationships: 50°C - (-10°C) = 40°C has meaning in arithmetic
- Ordinal assumptions: The model might assume equal "distance" between temperature steps

**One-Hot Encoding Solution:**

python

```python
from sklearn.feature_extraction import DictVectorizer

# Create temperature categorical data
temp_data = [
    {'temperature': '-10C', 'voltage': 3.2, 'current': 0.1},
    {'temperature': '0C', 'voltage': 3.3, 'current': 0.1},
    {'temperature': '25C', 'voltage': 3.4, 'current': 0.1},
    # ... more data points
]

vec = DictVectorizer(sparse=False, dtype=int)
encoded_features = vec.fit_transform(temp_data)

# Result: Each temperature becomes a separate binary column
# [temp=-10C, temp=0C, temp=25C, ..., voltage, current]
```

### 1.2 Step Index and Cycle Index as Categories

**Step Index Categories:**

- Step 1: Initial setup
- Step 2: Current application
- Step 3: OCV measurement
- Steps 5-8: Various test phases

**Cycle Index Categories:**

- Represents different test cycles within the same temperature condition
- Should be treated categorically as each cycle may have different characteristics

### 1.3 Practical Implementation

python

```python
# Temperature color mapping for visualization consistency
temp_colors = {
    '-10': '#0033A0',  # Deep blue
    '0': '#0066CC',    # Blue
    '10': '#3399FF',   # Light blue
    '20': '#66CC00',   # Green
    '25': '#FFCC00',   # Yellow (room temperature)
    '30': '#FF9900',   # Orange
    '40': '#FF6600',   # Dark orange
    '50': '#CC0000'    # Red
}

# One-hot encoding for temperature
temperature_encoded = pd.get_dummies(combined_data['temperature'], prefix='temp')
# Results in: temp_-10, temp_0, temp_10, temp_20, temp_25, temp_30, temp_40, temp_50
```

---

# 2. Derived Features

## 2.1 Time-Based Features

Battery performance is highly dependent on time-related patterns. We can extract meaningful temporal features:

**Polynomial Time Features:**

python

```python
from sklearn.preprocessing import PolynomialFeatures

# Create polynomial features from test time
poly = PolynomialFeatures(degree=3, include_bias=False)
time_features = poly.fit_transform(data[['Test_Time(s)']])

# Results in: [time, time², time³]
# Captures non-linear aging effects over test duration
```

**Derived Temporal Features:**

```python
# Time-based feature engineering
data['Test_Hours'] = data['Test_Time(s)'] / 3600
data['Test_Days'] = data['Test_Time(s)'] / (3600 * 24)
data['Time_Since_Cycle_Start'] = data.groupby('Cycle_Index')['Test_Time(s)'].transform(
    lambda x: x - x.min()
)
```

## 2.2 Electrical Performance Features

### Power and Energy Derivatives:

```python
# Power calculations
data['Instantaneous_Power'] = data['Voltage(V)'] * data['Current(A)']

# Energy efficiency features
data['Charge_Efficiency'] = (
    data['Discharge_Energy(Wh)'] / data['Charge_Energy(Wh)']
).fillna(0)

# Capacity utilization
data['Capacity_Utilization'] = (
    data['Discharge_Capacity(Ah)'] / data['Charge_Capacity(Ah)']
).fillna(0)
```

### Polynomial Features for Voltage-Current Relationships:

```python
# Capture non-linear V-I characteristics
voltage_current_poly = PolynomialFeatures(degree=2, include_bias=False)
vi_features = voltage_current_poly.fit_transform(
    data[['Voltage(V)', 'Current(A)']]
)

# Results in: [V, I, V², V*I, I²]
# Captures battery impedance characteristics and non-linearities
```

## 2.3 Temperature-Dependent Features

### Temperature Interaction Features:

```python
# Create features that capture temperature effects
data['Voltage_per_Temp'] = data['Voltage(V)'] / (data['Temperature(C)_1'] + 273.15)
data['Resistance_Temp_Factor'] = (
    data['Internal_Resistance(Ohm)'] * (data['Temperature(C)_1'] + 273.15)
)

# Arrhenius-like features for battery kinetics
data['Temp_Reciprocal'] = 1 / (data['Temperature(C)_1'] + 273.15)
```

### 2.4 Rate-Based Features

**Voltage and Current Rate Features:**

```python
# Rate of change features
data['Voltage_Rate'] = data['dV/dt(V/s)']  # Already provided
data['Current_Rate'] = data.groupby('Cycle_Index')['Current(A)'].diff() / \
                       data.groupby('Cycle_Index')['Test_Time(s)'].diff()

# Acceleration features (second derivatives)
data['Voltage_Acceleration'] = data.groupby('Cycle_Index')['dV/dt(V/s)'].diff() / \
                               data.groupby('Cycle_Index')['Test_Time(s)'].diff()
```

---

## 3. Text Features (Metadata Processing)

While our dataset is primarily numerical, we can apply text processing concepts to categorical metadata:

### 3.1 Test Phase Description Encoding

python

```python
# Simulated test phase descriptions based on Step_Index
step_descriptions = {
    1: "initial setup phase",
    2: "current application phase",
    3: "ocv measurement phase",
    5: "charge cycle phase",
    7: "discharge cycle phase",
    8: "final measurement phase"
}

# Convert to text corpus
phase_texts = [step_descriptions.get(step, "unknown phase")
               for step in data['Step_Index']]

# Apply TF-IDF vectorization
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
phase_features = tfidf.fit_transform(phase_texts)

# Results in features like: 'initial', 'setup', 'phase', 'current', etc.
```

---

## 4. Imputation of Missing Data

### 4.1 Handling Missing Temperature Data

Some datasets might have missing temperature readings:

python

```python
from sklearn.impute import SimpleImputer
import numpy as np

# Simulate missing data scenario
data_with_missing = data.copy()
missing_mask = np.random.random(len(data)) < 0.05  # 5% missing
data_with_missing.loc[missing_mask, 'Temperature(C)_1'] = np.nan

# Simple imputation strategies
mean_imputer = SimpleImputer(strategy='mean')
median_imputer = SimpleImputer(strategy='median')

# For temperature, median might be more robust
temp_imputed = median_imputer.fit_transform(
    data_with_missing[['Temperature(C)_1']].values
)
```

### 4.2 Advanced Imputation for Battery Data

python

```python
# Forward-fill for time series continuity
data['Voltage_Filled'] = data.groupby(['Temperature_Group', 'Cycle_Index'])\
                              ['Voltage(V)'].fillna(method='ffill')

# Interpolation for smooth electrical measurements
data['Current_Interpolated'] = data.groupby('Cycle_Index')['Current(A)']\
                                    .interpolate(method='linear')
```

---

## 5. Feature Pipelines

### 5.1 Complete Battery Analysis Pipeline

python

```python
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# Define feature groups
numerical_features = [
    'Voltage(V)', 'Current(A)', 'Test_Time(s)',
    'Internal_Resistance(Ohm)', 'Temperature(C)_1'
]

categorical_features = ['Step_Index', 'Cycle_Index']

# Create preprocessing pipeline
preprocessing_pipeline = ColumnTransformer([
    # Numerical features: impute, polynomial, scale
    ('num', Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('poly', PolynomialFeatures(degree=2, include_bias=False)),
        ('scaler', StandardScaler())
    ]), numerical_features),

    # Categorical features: one-hot encode
    ('cat', Pipeline([
        ('imputer', SimpleImputer(strategy='constant', fill_value='unknown')),
        ('onehot', OneHotEncoder(drop='first', sparse=False))
    ]), categorical_features)
])

# Complete pipeline with model
complete_pipeline = Pipeline([
    ('preprocessor', preprocessing_pipeline),
    ('regressor', LinearRegression())  # Example: predicting voltage degradation
])
```

## 5.2 Temperature-Specific Feature Pipeline

python

```python
# Specialized pipeline for temperature analysis
temp_analysis_pipeline = make_pipeline(
    # Step 1: Handle missing values
    SimpleImputer(strategy='mean'),

    # Step 2: Create polynomial features for non-linear temperature effects
    PolynomialFeatures(degree=3, interaction_only=True),

    # Step 3: Scale features
    StandardScaler(),

    # Step 4: Model (example: temperature effect on capacity)
    LinearRegression()
)

# Usage example
X = data[['Temperature(C)_1', 'Voltage(V)', 'Current(A)']]
y = data['Discharge_Capacity(Ah)']

temp_analysis_pipeline.fit(X, y)
predictions = temp_analysis_pipeline.predict(X)
```

---

# 6. Advanced Feature Engineering for Battery Applications

## 6.1 State of Health (SOH) Features

python

```python
# Capacity fade features
data['Capacity_Fade_Rate'] = data.groupby('Temperature_Group')\
                                 ['Discharge_Capacity(Ah)'].pct_change()

# Energy efficiency degradation
data['Efficiency_Trend'] = data.groupby(['Temperature_Group', 'Cycle_Index'])\
                               ['Charge_Efficiency'].transform(
                                   lambda x: x - x.iloc[0]
                               )
```

## 6.2 Thermal Management Features

```python
# Temperature differential features
data['Temp_Gradient'] = data['Temperature(C)_1'] - data['Temperature(C)_2']
data['Temp_Stability'] = data.groupby('Cycle_Index')['Temperature(C)_1'].std()

# Thermal resistance features
data['Thermal_Resistance'] = (
    (data['Temperature(C)_1'] - 25) / data['Instantaneous_Power']
).replace([np.inf, -np.inf], np.nan)
```

---

# 7. Feature Selection and Validation

## 7.1 Feature Importance Analysis

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_regression

# Random Forest for feature importance
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_processed, y)

feature_importance = pd.DataFrame({
    'feature': feature_names,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)

# Statistical feature selection
selector = SelectKBest(score_func=f_regression, k=10)
X_selected = selector.fit_transform(X_processed, y)
```

## 7.2 Cross-Temperature Validation

python

```python
# Validate features across different temperatures
from sklearn.model_selection import GroupKFold

# Use temperature as grouping variable for validation
groups = data['Temperature_Group'].map({
    temp: i for i, temp in enumerate(sorted(data['Temperature_Group'].unique()))
})

cv = GroupKFold(n_splits=5)
scores = cross_val_score(complete_pipeline, X, y, cv=cv, groups=groups)
```

---

## 8. Practical Implementation Example

### 8.1 Complete Feature Engineering Workflow

python

```python
def engineer_battery_features(data):
    """
    Complete feature engineering pipeline for battery OCV data
    """
    engineered_data = data.copy()

    # 1. Categorical encoding
    temp_dummies = pd.get_dummies(engineered_data['Temperature_Group'],
                                  prefix='temp')
    step_dummies = pd.get_dummies(engineered_data['Step_Index'],
                                  prefix='step')

    # 2. Derived features
    engineered_data['Power'] = (engineered_data['Voltage(V)'] *
                               engineered_data['Current(A)'])
    engineered_data['Energy_Efficiency'] = (
        engineered_data['Discharge_Energy(Wh)'] /
        engineered_data['Charge_Energy(Wh)']
    ).fillna(0)

    # 3. Polynomial features for key relationships
    poly_features = ['Voltage(V)', 'Current(A)', 'Test_Time(s)']
    poly = PolynomialFeatures(degree=2, include_bias=False)
    poly_array = poly.fit_transform(engineered_data[poly_features])
    poly_df = pd.DataFrame(poly_array,
                          columns=poly.get_feature_names_out(poly_features))

    # 4. Time-based features
    engineered_data['Test_Hours'] = engineered_data['Test_Time(s)'] / 3600
    engineered_data['Cycle_Duration'] = engineered_data.groupby('Cycle_Index')\
                                        ['Test_Time(s)']\
                                        .transform('max')

    # 5. Combine all features
    final_features = pd.concat([
        engineered_data,
        temp_dummies,
        step_dummies,
        poly_df
    ], axis=1)

    return final_features

# Apply to all temperature datasets
engineered_datasets = {}
for temp, dataset in temp_datasets.items():
```

```
    dataset['Temperature_Group'] = temp
    engineered_datasets[temp] = engineer_battery_features(dataset)
```

---

# 9. Visualization and Analysis

## 9.1 Feature Correlation Analysis

python

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Correlation heatmap of engineered features
key_features = [
    'Voltage(V)', 'Current(A)', 'Power', 'Energy_Efficiency',
    'Test_Hours', 'Internal_Resistance(Ohm)', 'Temperature(C)_1'
]

correlation_matrix = engineered_data[key_features].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Feature Correlation Matrix - Battery OCV Analysis')
plt.tight_layout()
```

## 9.2 Temperature Effect Visualization

python

```python
# Visualize temperature effects on key features
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

features_to_plot = ['Voltage(V)', 'Internal_Resistance(Ohm)',
                   'Energy_Efficiency', 'Discharge_Capacity(Ah)']

for i, feature in enumerate(features_to_plot):
    ax = axes[i//2, i%2]

    for temp, color in temp_colors.items():
        temp_data = engineered_datasets[f'{int(temp)}°C']
        ax.scatter(temp_data['Test_Hours'], temp_data[feature],
                  c=color, label=f'{temp}°C', alpha=0.6, s=1)

    ax.set_xlabel('Test Hours')
    ax.set_ylabel(feature)
    ax.set_title(f'{feature} vs Temperature')
    ax.legend()

plt.tight_layout()
plt.suptitle('Temperature Effects on Battery Performance Features',
             fontsize=16, y=1.02)
```

---

## 10. Conclusion

This comprehensive feature engineering demonstration shows how to transform raw battery OCV data into meaningful features for machine learning analysis. Key takeaways:

1. **Categorical Features**: Temperature and test phases should be one-hot encoded rather than treated as numerical values
2. **Derived Features**: Polynomial features capture non-linear battery behavior, while time-based features reveal aging effects
3. **Domain Knowledge**: Battery-specific features like power, efficiency, and thermal effects provide crucial insights
4. **Pipelines**: Systematic preprocessing ensures consistent and reproducible feature engineering
5. **Validation**: Cross-temperature validation ensures features generalize across operating conditions

The engineered features enable:

- **Predictive Modeling**: Battery lifetime and performance prediction
- **Anomaly Detection**: Identifying unusual battery behavior

- **Optimization**: Understanding optimal operating conditions
- **Quality Control**: Monitoring battery manufacturing consistency

This approach can be extended to other battery datasets and adapted for different battery chemistries and applications.

---

## References


- Scikit-Learn Documentation: Feature Engineering and Preprocessing
- Battery Testing Standards: IEC 61960, IEEE 1625
- A123 Systems Battery Technology Documentation
- Machine Learning for Battery Applications: State-of-the-Art Review