Feature Engineering & Selection

From Raw Data to ML-Ready Features

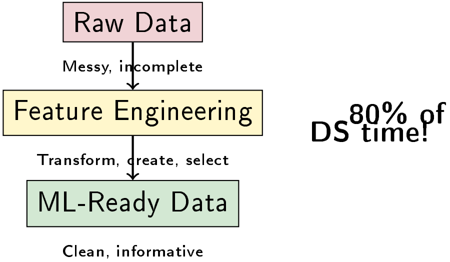
Diogo Ribeiro
ESMAD – Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

November 16, 2025

Outline

The Art and Science of Feature Engineering

> *"Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering."*
>
> – *Andrew Ng*



Why Features Matter:

- Garbage in, garbage out: Poor features ⇒ poor models
- Domain knowledge: Good features encode expert insights
- Model performance: Often more impact than algorithm choice
- Interpretability: Good features are meaningful to humans

The 80-20 Rule

80% of data science time is spent on data preparation and feature engineering, 20% on modeling.

Feature Engineering Pipeline Overview

The Complete Pipeline:

1. **Data Understanding**
   - Exploratory data analysis
   - Data quality assessment
   - Domain knowledge integration
2. **Cleaning & Preprocessing**
   - Missing value handling
   - Outlier detection/treatment
   - Data type conversions
3. **Feature Creation**
   - Transformations
   - Interactions
   - Domain-specific features
4. **Feature Selection**
   - Remove redundant features
   - Statistical significance

Success Metrics

- **Model Performance**: Accuracy, AUC, RMSE
- **Interpretability**: Can humans understand features?
- **Stability**: Robust to new data
- **Efficiency**: Fast to compute and store

Common Mistakes

- Data leakage: Using future information
- Overfitting: Too many features for sample size
- Domain ignorance: Features that don't make sense

Understanding Your Data Types

Data Type	Examples	Challenges	Common Transformations
Numerical	Age, income, temperature	Skewness, outliers, scale	Log, square root, standardization
Categorical	Color, country, brand	High cardinality, ordering	One-hot, label encoding, embeddings
Ordinal	Education level, ratings	Preserving order	Ordinal encoding, polynomial features
Temporal	Timestamps, dates	Seasonality, trends	Date parts, lags, rolling statistics
Text	Reviews, descriptions	High dimensionality	TF-IDF, embeddings, sentiment
Geospatial	Coordinates, addresses	Projection, distance	Distance features, clustering

Numerical Feature Transformations

Common Issues with Numerical Features:

```
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.preprocessing import (
    StandardScaler, MinMaxScaler,
    RobustScaler, PowerTransformer,
    QuantileTransformer
)

# Example: Highly skewed income data
np.random.seed(42)
income = np.random.lognormal(10, 1, 1000)
print(f"Skewness: {stats.skew(income):.2f}")
print(f"Range: [{income.min():.0f}, {income.
    max():.0f}]")

# Transformation strategies
transformations = {
```

Scaling Strategies:

- **StandardScaler**: $z = \frac{x-\mu}{\sigma}$
  - Good for normally distributed data
  - Sensitive to outliers
- **MinMaxScaler**: $x' = \frac{x-\min(x)}{\max(x)-\min(x)}$
  - Bounds data to [0,1]
  - Very sensitive to outliers
- **RobustScaler**: Uses median and IQR

Categorical Feature Encoding

The Categorical Challenge: ML algorithms need numbers, not categories.

```
import pandas as pd
from sklearn.preprocessing import (
    LabelEncoder, OneHotEncoder,
    OrdinalEncoder
)
from category_encoders import (
    TargetEncoder, BinaryEncoder,
    HashingEncoder, LeaveOneOutEncoder
)

# Sample categorical data
data = pd.DataFrame({
    'color': ['red', 'blue', 'green', 'red', 'blue
    '],
    'size': ['small', 'medium', 'large', 'medium',
     'small'],
    'brand': ['nike', 'adidas', 'puma', 'nike', '
```

Encoding Strategy Guide:

Method	Cardinality	Best For
Label	Any	Ordinal data
One-Hot	Low (< 10)	Nominal data
Target	Medium	High predictive power

Temporal Feature Engineering

Time Series Features: Extract meaningful patterns from timestamps.

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Sample time series data
dates = pd.date_range(start='2020-01-01', end='
    2023-12-31', freq='D')
np.random.seed(42)
sales = 100 + 10*np.sin(2*np.pi*np.arange(len(
    dates))/365.25) + \
        5*np.random.randn(len(dates))

df = pd.DataFrame({'date': dates, 'sales': sales})
df['date'] = pd.to_datetime(df['date'])

# Basic temporal features
df['year'] = df['date'].dt.year
```

Temporal Feature Categories:

- **Date Parts**: Year, month, day, hour
- **Cyclical**: Sin/cos encoding for

Polynomial and Interaction Features

Capturing Non-linear Relationships:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import
    PolynomialFeatures
from sklearn.model_selection import
    cross_val_score
from sklearn.linear_model import
    LinearRegression
from sklearn.pipeline import Pipeline

# Generate sample data with interactions
np.random.seed(42)
n_samples = 1000
X1 = np.random.randn(n_samples)
X2 = np.random.randn(n_samples)
X3 = np.random.randn(n_samples)
```

When to Use Polynomial Features:

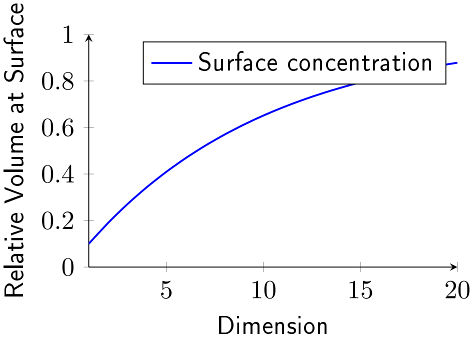
- Linear models: Add non-linearity
- Known relationships: Domain knowledge

Case Study: E-commerce Customer Features

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Sample e-commerce transaction data
np.random.seed(42)
transactions = pd.DataFrame({
    'customer_id': np.repeat(range(1000), 10),
    'transaction_date': pd.to_datetime('2023-01-01
    ') +
                          pd.to_timedelta(np.random.
    randint(0, 365, 10000), unit='D'),
    'amount': np.random.exponential(50, 10000),
    'product_category': np.random.choice(['
    electronics', 'clothing', 'books', 'home'],
    10000),
    'is_returned': np.random.binomial(1, 0.1,
```

Why High Dimensions Are Problematic:

- **Sparsity**: Data points become sparse in high-D space
- **Distance concentration**: All points equidistant
- **Overfitting**: More parameters than samples
- **Computational cost**: $O(d^k)$ complexity
- **Visualization**: Impossible to plot > 3D



Mathematical Intuition: Volume of unit hypersphere in d dimensions:

$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

The $d \gg n$ Problem

When dimensions exceed samples:

- Perfect separation possible

Principal Component Analysis (PCA)

Goal: Find linear combinations of features that maximize variance.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import
    load_breast_cancer
from sklearn.preprocessing import
    StandardScaler

# Load high-dimensional dataset
data = load_breast_cancer()
X, y = data.data, data.target

print(f"Original dimensions: {X.shape}")

# Standardize features (crucial for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Mathematical Formulation:

$$\text{Maximize:} \quad \text{Var}(\mathbf{w}^T \mathbf{X}) \tag{3}$$

$$\text{Subject to:} \quad \|\mathbf{w}\|^2 = 1 \tag{4}$$

Non-linear Methods for Complex Data:

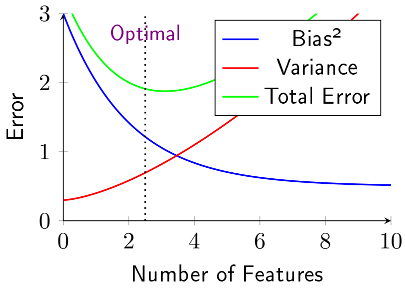
```
import numpy as np
from sklearn.manifold import TSNE
from umap import UMAP
from sklearn.decomposition import KernelPCA
from sklearn.datasets import make_swiss_roll

# Generate non-linear data
X, color = make_swiss_roll(n_samples=1000, noise
    =0.1)

# 1. Kernel PCA (non-linear PCA)
kpca_rbf = KernelPCA(n_components=2, kernel='rbf',
     gamma=0.1)
X_kpca = kpca_rbf.fit_transform(X)

# 2. t-SNE (preserves local structure)
tsne = TSNE(n_components=2, perplexity=30,
```

Method Comparison:

Feature Selection: Why and When

Why Remove Features?

- **Overfitting**: Too many features relative to samples
- **Noise**: Irrelevant features hurt performance
- **Multicollinearity**: Redundant information
- **Computational cost**: Storage and processing
- **Interpretability**: Simpler models are easier to understand



The Bias-Variance Perspective:

$$\text{Fewer features} \Rightarrow \text{Higher bias, Lower variance} \tag{5}$$

When to Apply Feature Selection

- High-dimensional data ($p \gg n$)
- Noisy or redundant features
- Interpretability requirements

Filter Methods: Statistical Feature Selection

Idea: Rank features by statistical measures, independent of the learning algorithm.

```python
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.feature_selection import (
    SelectKBest, f_classif, chi2,
    mutual_info_classif,
    VarianceThreshold,
    SelectPercentile
)
from sklearn.preprocessing import StandardScaler
from scipy.stats import pearsonr

# Generate sample data with irrelevant features
X, y = make_classification(
    n_samples=1000, n_features=20,
    n_informative=5, n_redundant=3,
    n_clusters_per_class=1, random_state=42
)
```

Wrapper Methods: Model-Based Selection

Idea: Use the target model's performance to evaluate feature subsets.

```
import numpy as np
from sklearn.model_selection import
    cross_val_score
from sklearn.ensemble import
    RandomForestClassifier
from sklearn.feature_selection import (
    RFE, RFECV, SequentialFeatureSelector
)
from sklearn.linear_model import
    LogisticRegression
from sklearn.datasets import make_classification

# Generate data
X, y = make_classification(
    n_samples=500, n_features=15,
    n_informative=5, n_redundant=3,
    random_state=42
```

Embedded Methods: Built-in Feature Selection

Idea: Feature selection is integrated into the model training process.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.linear_model import Lasso, Ridge,
    ElasticNet
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import
    SelectFromModel
from sklearn.model_selection import
    cross_val_score

# Generate regression data with some irrelevant
    features
X, y = make_regression(
    n_samples=200, n_features=20,
    n_informative=5, noise=10,
    random_state=42
)
```

Data Leakage: The Silent Killer

What is Data Leakage? Information from the future or target variable inadvertently included in features.

Types of Leakage:

- **Temporal leakage**: Using future information
- **Target leakage**: Features that contain target info
- **Train-test leakage**: Data preprocessing on full dataset

Common Examples:

- Credit scoring using payment history after loan decision
- Medical diagnosis using treatment

Preventing Leakage:

1. **Time-aware splits**:
   - Train on past, test on future
   - No random shuffling for time series
2. **Proper preprocessing**:
   - Fit transformers only on training data
   - Apply to test data, don't refit
3. **Domain knowledge**:
   - What information is available when?
   - Business process understanding
4. **Feature audit**:
   - Check correlations with target
   - Validate with domain experts

Golden Rule

Pipeline Design and Reproducibility

Best Practice: Use sklearn pipelines for reproducible feature engineering.

```
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import (
    StandardScaler, OneHotEncoder,
    FunctionTransformer
)
from sklearn.feature_selection import SelectKBest,
    f_regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

# Sample mixed-type dataset
np.random.seed(42)
data = pd.DataFrame({
    'numeric1': np.random.randn(1000),
```

Feature Engineering Checklist

Data Understanding

- ☐ Exploratory data analysis
- ☐ Missing value patterns
- ☐ Outlier detection
- ☐ Data type validation
- ☐ Domain expert consultation

Feature Creation

- ☐ Handle missing values appropriately
- ☐ Scale/normalize numerical features
- ☐ Encode categorical variables
- ☐ Create interaction features
- ☐ Extract temporal features
- ☐ Engineer domain-specific features

Leakage Prevention

- ☐ Time-aware data splitting
- ☐ Fit transformers only on training data
- ☐ Audit features for target information
- ☐ Validate business logic
- ☐ Check correlation with target

Validation & Testing

- ☐ Cross-validation with proper splits
- ☐ Test on unseen data
- ☐ Monitor feature distributions
- ☐ A/B test in production
- ☐ Track model performance over time

Key Takeaways

Core Principles:

- **Domain knowledge** is as important as technical skills
- **Systematic approach** beats ad-hoc feature creation
- **Validation** is crucial for avoiding overfitting
- **Pipelines** ensure reproducibility and prevent leakage

Common Pitfalls Avoided:

- Data leakage through improper preprocessing
- Overfitting with too many features
- Scale issues in mixed-type data
- Target leakage in feature creation
- Irreproducible results

Technical Skills Learned:

- Data type-specific transformations
- Categorical encoding strategies
- Dimensionality reduction techniques
- Feature selection methods

The Art vs Science

Science: Statistical methods, validation procedures, systematic evaluation
Art: Domain insights, creative feature combinations, business intuition

Next Steps in Your Data Science Journey

Immediate Next Topics:

1. **Causal Inference for Data Scientists**
    - Moving beyond correlation
    - Experimental design
    - Observational causal methods
2. **Explainable AI & Model Interpretability**
    - SHAP and LIME
    - Global vs local explanations
    - Building trust in models
3. **Experimental Design & A/B Testing**

Practice Projects:

- Build end-to-end feature engineering pipeline
- Kaggle competition with focus on feature engineering
- Industry-specific feature creation
- Automated feature engineering tools

Advanced Topics to Explore:

- Automated feature engineering (Featuretools)
- Deep feature synthesis
- Graph-based features
- Text feature engineering

Thank You

Questions & Discussion

Diogo Ribeiro
ESMAD – Escola Superior de Média Arte e Design
Lead Data Scientist, Mysense.ai

`dfr@esmad.ipp.pt`
`https://orcid.org/0009-0001-2022-7072`

Slides and code available at:
`github.com/diogoribeiro7/academic-presentations`

Next: Causal Inference for Data Scientists