Feature Engineering

Guida Completa all'Ingegneria delle Funzionalità

nel Machine Learning e Data Analysis
Basato sull'analisi del blog DataMasters.it
Integrato con ricerca aggiuntiva

4 settembre 2025

Indice

1 Introduzione al Feature Engineering **4**
1.1 Definizione e Concetti Fondamentali 4
1.1.1 Cosa sono le Features 4
1.1.2 Obiettivi del Feature Engineering 4

2 Importanza del Feature Engineering **6**
2.1 Il Valore delle Feature nei Modelli Predittivi 6
2.1.1 Esempio Pratico: Previsione delle Vendite 6
2.2 Come il Feature Engineering Migliora le Prestazioni 6
2.2.1 Semplificazione del Problema 7
2.2.2 Riduzione della Dimensionalità 7
2.2.3 Incorporazione di Conoscenza del Dominio 7
2.3 Riduzione del Rumore nei Dati 7
2.3.1 Strategie per la Gestione del Rumore 7

3 Tecniche Principali di Feature Engineering **9**
3.1 Trasformazioni Matematiche 9
3.1.1 Trasformazioni Logaritmiche 9
3.1.2 Trasformazioni di Potenza 9
3.1.3 Trasformazioni Radice 9
3.2 Gestione delle Variabili Categoriche 10
3.2.1 One-Hot Encoding 10
3.2.2 Label Encoding 10
3.2.3 Target Encoding 10
3.3 Feature Scaling e Normalizzazione 10
3.3.1 Min-Max Scaling 10
3.3.2 Standardizzazione (Z-score) 10
3.3.3 Robust Scaling 11
3.4 Creazione di Feature Interattive 11
3.4.1 Feature Polinomiali 11
3.4.2 Feature Razionali 11
3.4.3 Binning 11

4 Gestione dei Valori Mancanti **12**
4.1 Strategie di Imputazione 12
4.1.1 Imputazione Semplice 12
4.1.2 Imputazione Avanzata 12
4.1.3 Indicatori di Mancanza 12

5 Feature Engineering per Dati Temporali **13**
5.1 Estrazione di Componenti Temporali . . . 13
5.1.1 Componenti Base . . . 13
5.1.2 Feature Ciclici . . . 13
5.1.3 Aggregazioni Temporali . . . 13

6 Feature Engineering per Testi **15**
6.1 Preprocessing del Testo . . . 15
6.1.1 Operazioni di Base . . . 15
6.2 Tecniche di Vettorizzazione . . . 15
6.2.1 Bag of Words . . . 15
6.2.2 TF-IDF . . . 15
6.2.3 N-grammi . . . 16
6.3 Feature Testuali Avanzate . . . 16
6.3.1 Statistiche del Testo . . . 16
6.3.2 Feature Semantiche . . . 16

7 Selezione delle Feature **17**
7.1 Differenza tra Feature Engineering e Feature Selection . . . 17
7.1.1 Metodi di Feature Selection . . . 17
7.2 Valutazione dell'Importanza delle Feature . . . 18
7.2.1 Feature Importance da Modelli Tree-based . . . 18
7.2.2 Permutation Importance . . . 18

8 Automazione del Feature Engineering **20**
8.1 Strumenti di Automazione . . . 20
8.1.1 Featuretools . . . 20
8.1.2 TPOT . . . 21
8.2 AutoML e Feature Engineering . . . 22
8.2.1 Vantaggi dell'Automazione . . . 22
8.2.2 Limitazioni . . . 22

9 Competenze e Strumenti per Feature Engineer **23**
9.1 Percorso Formativo . . . 23
9.1.1 Competenze Fondamentali . . . 23
9.2 Librerie e Strumenti Essenziali . . . 24
9.2.1 Ecosistema Python . . . 24
9.2.2 Ecosistema R . . . 24
9.3 Ambiente di Sviluppo . . . 25
9.3.1 IDE e Notebook . . . 25
9.3.2 Controllo Versione . . . 25
9.3.3 Deployment e Produzione . . . 25

10 Ruolo del Feature Engineer nel Team **26**
10.1 Responsabilità e Collaborazioni . . . 26
10.1.1 Collaborazione con Data Engineer . . . 26
10.1.2 Collaborazione con Data Scientist . . . 26
10.1.3 Collaborazione con Business Stakeholder . . . 26
10.2 Attività Quotidiane . . . 26

10.2.1 Sviluppo e Manutenzione 26
10.2.2 Ricerca e Sperimentazione 27
10.2.3 Governance e Qualità 27

11 Casi Studio Pratici **28**
11.1 Caso Studio 1: E-commerce - Previsione Abbandono Carrello 28
11.1.1 Contesto del Problema 28
11.1.2 Dataset Originale 28
11.1.3 Feature Engineering Implementate 28
11.1.4 Risultati 29
11.2 Caso Studio 2: Fintech - Credit Scoring 30
11.2.1 Contesto del Problema 30
11.2.2 Feature Engineering per Dati Finanziari 30
11.2.3 Risultati 31
11.3 Caso Studio 3: Healthcare - Predizione Readmission 31
11.3.1 Feature Engineering per Dati Medici 31

12 Best Practices e Considerazioni Avanzate **33**
12.1 Pipeline di Feature Engineering 33
12.1.1 Organizzazione del Codice 33
12.1.2 Gestione della Data Leakage 34
12.2 Validazione e Testing 34
12.2.1 Unit Testing per Feature Engineering 34
12.2.2 Cross-Validation Consapevole del Tempo 35
12.3 Monitoraggio in Produzione 36
12.3.1 Feature Drift Detection 36
12.3.2 Performance Monitoring 37

13 Tendenze Future e Tecnologie Emergenti **39**
13.1 Deep Learning e Feature Learning 39
13.1.1 Automated Feature Extraction 39
13.2 Quantum Feature Engineering 40
13.2.1 Quantum Computing per Ottimizzazione 40
13.3 Edge Computing e Real-time Features 40
13.3.1 Feature Engineering Distribuita 40
13.4 Explainable AI e Feature Interpretation 41
13.4.1 SHAP Values per Feature Analysis 41

14 Conclusioni **43**
14.1 Riepilogo dei Concetti Chiave 43
14.1.1 Fattori Critici di Successo 43
14.2 Impatto sul Business 43
14.3 Raccomandazioni per il Futuro 44
14.3.1 Per i Professionisti 44
14.3.2 Per le Organizzazioni 44
14.4 Tendenze Emergenti 44
14.5 Considerazioni Finali 44

Capitolo 1

Introduzione al Feature Engineering

1.1 Definizione e Concetti Fondamentali

Il **feature engineering** rappresenta uno dei pilastri fondamentali nel campo dell'analisi dei dati e del machine learning. Questo processo, spesso sottovalutato ma cruciale, trasforma i dati grezzi in caratteristiche significative che possono determinare il successo o il fallimento di un modello predittivo.

L'ingegneria delle funzionalità, o feature engineering, è il processo attraverso il quale i dati grezzi vengono trasformati in caratteristiche (features) che rappresentano meglio il problema sottostante che si sta cercando di risolvere. Questo processo richiede una profonda comprensione del dominio specifico, creatività e competenze tecniche avanzate.

1.1.1 Cosa sono le Features

Una *feature* è qualsiasi proprietà misurabile di un dato che può essere utilizzata come input per algoritmi di machine learning. Le features possono essere:

- **Numeriche**: rappresentano dati continui quantitativi (es. età, altezza, reddito)
- **Categoriche**: contengono solo caratteristiche discrete (es. genere, città di residenza)
  - *Binarie*: due possibili categorie
  - *Multi-classe*: multiple categorie
- **Testuali**: contengono dati di testo (es. recensioni prodotti, descrizioni)

1.1.2 Obiettivi del Feature Engineering

Il feature engineering comprende diverse operazioni sofisticate:

1. Creazione di nuove caratteristiche attraverso la combinazione di quelle esistenti
2. Trasformazione delle variabili per ottimizzarne la distribuzione
3. Encoding delle variabili categoriche
4. Gestione accurata dei valori mancanti

5. Normalizzazione e standardizzazione dei dati

6. Estrazione di caratteristiche significative da dati non strutturati

Tutte queste operazioni vengono eseguite con l'obiettivo di creare un set di feature che possa rappresentare al meglio la complessità del problema da risolvere.

Capitolo 2

Importanza del Feature Engineering

2.1 Il Valore delle Feature nei Modelli Predittivi

L'importanza del feature engineering nel campo dell'analisi dei dati e del machine learning non può essere sottovalutata. Questa disciplina rappresenta spesso la differenza tra un modello mediocre e uno eccellente. La qualità delle feature utilizzate per addestrare un modello ha un impatto diretto sulle sue prestazioni.

I modelli predittivi basano le loro decisioni sulle caratteristiche che vengono loro fornite. Se queste caratteristiche non rappresentano adeguatamente il problema sottostante, il modello non potrà mai raggiungere prestazioni ottimali, indipendentemente dalla sua sofisticatezza.

2.1.1 Esempio Pratico: Previsione delle Vendite

Per illustrare l'importanza del feature engineering, consideriamo un problema di previsione delle vendite. Invece di utilizzare semplicemente la data come input, un buon feature engineering potrebbe estrarre:

- Giorno della settimana
- Mese dell'anno
- Stagione
- Indicatore di giorno festivo
- Distanza dal weekend
- Periodo dell'anno (inizio/metà/fine mese)

Queste nuove caratteristiche potrebbero catturare pattern stagionali e cicli di vendita che il modello potrebbe trovare difficile da apprendere dai dati grezzi.

2.2 Come il Feature Engineering Migliora le Prestazioni

Il feature engineering gioca un ruolo cruciale nel miglioramento delle prestazioni dei modelli attraverso diversi meccanismi:

2.2.1 Semplificazione del Problema

Creando feature più informative, il feature engineering può rendere più semplice per il modello apprendere le relazioni nei dati. Invece di dover scoprire pattern complessi da dati grezzi, il modello può lavorare con caratteristiche già significative dal punto di vista del dominio.

2.2.2 Riduzione della Dimensionalità

Attraverso la creazione di feature più significative, è possibile ridurre la dimensionalità dei dati mantenendo o addirittura aumentando il contenuto informativo. Questo può portare a:

- Modelli più efficienti
- Minore rischio di overfitting
- Tempi di addestramento ridotti
- Migliore interpretabilità

2.2.3 Incorporazione di Conoscenza del Dominio

Il feature engineering permette di incorporare la conoscenza specifica del dominio nel processo di modellazione. Questo è particolarmente importante in campi specializzati come:

- Finanza (ratios finanziari, indicatori tecnici)
- Medicina (biomarkers, indici clinici)
- Ingegneria (parametri fisici, coefficienti)
- Marketing (customer lifetime value, segmentazione)

2.3 Riduzione del Rumore nei Dati

Uno degli aspetti più significativi del feature engineering è la sua capacità di ridurre il rumore nei dati. I dati del mondo reale sono spesso caratterizzati da variazioni casuali che non contengono informazioni utili per il problema da risolvere.

2.3.1 Strategie per la Gestione del Rumore

Attraverso tecniche mirate di feature engineering, è possibile implementare strategie efficaci:

- **Rimozione degli outlier**: identificazione e gestione di valori anomali
- **Smoothing dei dati**: applicazione di filtri per ridurre variazioni casuali
- **Aggregazione**: combinazione di informazioni a livelli più appropriati

- **Winsorization**: limitazione di valori estremi
- **Capping**: applicazione di soglie massime e minime

Questo processo di raffinamento permette di creare feature più robuste e meno sensibili al rumore, consentendo ai modelli di concentrarsi sugli aspetti veramente rilevanti dei dati.

Capitolo 3

Tecniche Principali di Feature Engineering

3.1 Trasformazioni Matematiche

Le trasformazioni matematiche costituiscono una categoria fondamentale delle tecniche di feature engineering. Queste includono operazioni che possono aiutare a normalizzare le distribuzioni o linearizzare le relazioni tra variabili.

3.1.1 Trasformazioni Logaritmiche

$$y = \log(x + c) \tag{3.1}$$

dove c è una costante per gestire valori zero o negativi.
Applicazioni:

- Normalizzazione di distribuzioni fortemente asimmetriche
- Riduzione dell'impatto di outlier
- Linearizzazione di relazioni esponenziali

3.1.2 Trasformazioni di Potenza

La famiglia di trasformazioni Box-Cox:

$$y(\lambda) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{se } \lambda \neq 0 \\ \ln(x) & \text{se } \lambda = 0 \end{cases} \tag{3.2}$$

3.1.3 Trasformazioni Radice

$$y = \sqrt[n]{x} \tag{3.3}$$

Particolarmente utili per dati di conteggio o distribuzioni di Poisson.

3.2 Gestione delle Variabili Categoriche

3.2.1 One-Hot Encoding

Converte variabili categoriche in vettori binari:

Colore	Rosso	Verde	Blu
Rosso	1	0	0
Verde	0	1	0
Blu	0	0	1

Tabella 3.1: Esempio di One-Hot Encoding

3.2.2 Label Encoding

Assegna valori numerici ordinali alle categorie:

Livello	Codifica
Basso	0
Medio	1
Alto	2

Tabella 3.2: Esempio di Label Encoding

3.2.3 Target Encoding

Sostituisce le categorie con la media del target per quella categoria:

$$\text{encoded\_value} = \frac{\sum_{i \in \text{categoria}} y_i}{|\text{categoria}|} \tag{3.4}$$

3.3 Feature Scaling e Normalizzazione

3.3.1 Min-Max Scaling

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3.5}$$

Ridimensiona i valori nell'intervallo [0, 1].

3.3.2 Standardizzazione (Z-score)

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma} \tag{3.6}$$

dove μ è la media e σ è la deviazione standard.

3.3.3 Robust Scaling

$$x_{\text{robust}} = \frac{x - \text{mediana}}{Q_3 - Q_1} \tag{3.7}$$

Utilizza mediana e range interquartile, più robusto agli outlier.

3.4 Creazione di Feature Interattive

3.4.1 Feature Polinomiali

Creazione di termini di interazione tra features:

$$\text{feature\_interaction} = x_1 \times x_2 \tag{3.8}$$

3.4.2 Feature Razionali

$$\text{ratio\_feature} = \frac{x_1}{x_2} \tag{3.9}$$

Particolarmente utili in contesti finanziari e di business.

3.4.3 Binning

Conversione di variabili continue in categoriche:

```python
import pandas as pd
import numpy as np

# Creazione di bin per eta
def create_age_bins(age):
    if age < 18:
        return 'Minorenne'
    elif age < 30:
        return 'Giovane_Adulto'
    elif age < 50:
        return 'Adulto'
    else:
        return 'Senior'

df['age_group'] = df['age'].apply(create_age_bins)
```

Listing 3.1: Esempio di Binning in Python

Capitolo 4

Gestione dei Valori Mancanti

4.1 Strategie di Imputazione

La gestione dei valori mancanti è un aspetto cruciale del feature engineering. Diverse strategie possono essere applicate a seconda del tipo di dati e del contesto.

4.1.1 Imputazione Semplice

- **Media**: per variabili numeriche con distribuzione normale
- **Mediana**: per variabili numeriche con outlier
- **Moda**: per variabili categoriche
- **Valore costante**: quando ha senso dal punto di vista del dominio

4.1.2 Imputazione Avanzata

- **K-Nearest Neighbors (KNN)**: utilizza la similarità tra osservazioni
- **Regressione**: predice valori mancanti usando altre variabili
- **Multiple Imputation**: crea multiple versioni del dataset
- **MICE** (Multivariate Imputation by Chained Equations)

4.1.3 Indicatori di Mancanza

Creazione di flag binari per indicare la presenza di valori mancanti:

```
# Creazione di flag per valori mancanti
df['income_missing'] = df['income'].isnull().astype(int)

# Imputazione con media
df['income_filled'] = df['income'].fillna(df['income'].mean())
```

Listing 4.1: Creazione di indicatori di mancanza

Capitolo 5

Feature Engineering per Dati Temporali

5.1 Estrazione di Componenti Temporali

I dati temporali offrono ricche opportunità per il feature engineering:

5.1.1 Componenti Base

- Anno, mese, giorno
- Ora, minuto, secondo
- Giorno della settimana
- Giorno dell'anno
- Settimana dell'anno

5.1.2 Feature Ciclici

Per catturare la natura ciclica del tempo:

$$\begin{aligned} \text{hour\_sin} &= \sin\left(\frac{2\pi \times \text{hour}}{24}\right) \\ \text{hour\_cos} &= \cos\left(\frac{2\pi \times \text{hour}}{24}\right) \end{aligned} \tag{5.1}$$

5.1.3 Aggregazioni Temporali

- **Moving averages**: medie mobili su finestre temporali
- **Lag features**: valori di periodi precedenti
- **Lead features**: valori di periodi futuri
- **Differences**: variazioni tra periodi
- **Rate of change**: velocità di cambiamento

```python
import pandas as pd

# Conversione a datetime
df['date'] = pd.to_datetime(df['date'])

# Estrazione componenti temporali
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day_of_week'] = df['date'].dt.dayofweek
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)

# Feature ciclici
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)

# Moving averages
df['sales_ma_7'] = df['sales'].rolling(window=7).mean()
df['sales_ma_30'] = df['sales'].rolling(window=30).mean()

# Lag features
df['sales_lag_1'] = df['sales'].shift(1)
df['sales_lag_7'] = df['sales'].shift(7)
```

Listing 5.1: Feature temporali in Python

Capitolo 6

Feature Engineering per Testi

6.1 Preprocessing del Testo

6.1.1 Operazioni di Base

- Conversione in minuscolo
- Rimozione punteggiatura
- Rimozione stopwords
- Stemming e lemmatizzazione
- Tokenizzazione

6.2 Tecniche di Vettorizzazione

6.2.1 Bag of Words

Rappresentazione basata sulla frequenza delle parole:

Documento	”machine”	”learning”	”data”	”science”
Doc1	2	1	1	0
Doc2	1	1	2	1

Tabella 6.1: Esempio di Bag of Words

6.2.2 TF-IDF

Term Frequency-Inverse Document Frequency:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t) \tag{6.1}$$

dove:

$$\begin{aligned} \text{TF}(t, d) &= \frac{\text{frequenza di t in d}}{\text{numero totale di termini in d}} \\ \text{IDF}(t) &= \log\left(\frac{\text{numero totale di documenti}}{\text{numero di documenti contenenti t}}\right) \end{aligned} \tag{6.2}$$

6.2.3 N-grammi

Sequenze di n parole consecutive:

- **Unigrammi**: singole parole
- **Bigrammi**: coppie di parole
- **Trigrammi**: triple di parole

6.3 Feature Testuali Avanzate

6.3.1 Statistiche del Testo

- Lunghezza del testo
- Numero di parole
- Numero di frasi
- Lunghezza media delle parole
- Numero di caratteri maiuscoli
- Numero di caratteri speciali

6.3.2 Feature Semantiche

- Sentiment score
- Topic modeling
- Word embeddings (Word2Vec, GloVe)
- Document embeddings

Capitolo 7

Selezione delle Feature

7.1 Differenza tra Feature Engineering e Feature Selection

Mentre il feature engineering si concentra sulla *creazione* e trasformazione delle caratteristiche, la feature selection è un processo complementare che si occupa di *identificare* quali caratteristiche sono più rilevanti per il problema.

7.1.1 Metodi di Feature Selection

Metodi Filtro

Utilizzano misure statistiche per valutare la rilevanza:

- Correlazione di Pearson
- Chi-quadrato
- Mutual Information
- ANOVA F-test

Metodi Wrapper

Utilizzano algoritmi di machine learning per valutare subset di feature:

- Recursive Feature Elimination (RFE)
- Forward Selection
- Backward Elimination
- Genetic Algorithms

Metodi Embedded

Integrano la selezione nel processo di addestramento:

- LASSO Regression
- Ridge Regression
- Elastic Net
- Tree-based feature importance

7.2 Valutazione dell'Importanza delle Feature

7.2.1 Feature Importance da Modelli Tree-based

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
import pandas as pd

# Caricamento dati
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target

# Addestramento modello
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Estrazione feature importance
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)

print(feature_importance)
```

Listing 7.1: Feature importance con Random Forest

7.2.2 Permutation Importance

Misura la diminuzione delle prestazioni quando i valori di una feature vengono permutati:

```python
from sklearn.inspection import permutation_importance

# Calcolo permutation importance
perm_importance = permutation_importance(
    rf, X, y, n_repeats=10, random_state=42
)

```

```
importance_df = pd.DataFrame({
    'feature': X.columns,
    'importance': perm_importance.importances_mean,
    'std': perm_importance.importances_std
}).sort_values('importance', ascending=False)
```

Listing 7.2: Permutation importance

Capitolo 8

Automazione del Feature Engineering

8.1 Strumenti di Automazione

8.1.1 Featuretools

Libreria Python per automated feature engineering:

```
import featuretools as ft
import pandas as pd

# Creazione dataset esempio
customers = pd.DataFrame({
    'customer_id': [1, 2, 3],
    'age': [25, 30, 35],
    'location': ['NY', 'CA', 'TX']
})

orders = pd.DataFrame({
    'order_id': [1, 2, 3, 4],
    'customer_id': [1, 1, 2, 3],
    'amount': [100, 200, 150, 300],
    'date': pd.to_datetime(['2023-01-01', '2023-01-15',
                            '2023-02-01', '2023-02-15'])
})

# Creazione EntitySet
es = ft.EntitySet(id="sales_data")
es = es.add_dataframe(
    dataframe_name="customers",
    dataframe=customers,
    index="customer_id"
)
es = es.add_dataframe(
    dataframe_name="orders",
    dataframe=orders,
    index="order_id",
    time_index="date"
```

```
)

# Definizione relazioni
es = es.add_relationship("customers", "customer_id",
                         "orders", "customer_id")

# Generazione automatica features
feature_matrix, feature_defs = ft.dfs(
    entityset=es,
    target_dataframe_name="customers",
    max_depth=2
)
```

Listing 8.1: Esempio con Featuretools

8.1.2 TPOT

Utilizza algoritmi genetici per ottimizzare pipeline di machine learning:

```
from tpot import TPOTClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

# Caricamento dati
digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, train_size=0.75, test_size=0.25
)

# Configurazione TPOT
tpot = TPOTClassifier(
    generations=5,
    population_size=20,
    verbosity=2,
    random_state=42
)

# Ricerca automatica della migliore pipeline
tpot.fit(X_train, y_train)
print(f"Accuracy: {tpot.score(X_test, y_test)}")

# Esportazione della pipeline ottimale
tpot.export('tpot_pipeline.py')
```

Listing 8.2: Esempio con TPOT

8.2 AutoML e Feature Engineering

8.2.1 Vantaggi dell'Automazione

- Riduzione del tempo di sviluppo
- Esplorazione sistematica dello spazio delle feature
- Riduzione degli errori umani
- Scalabilità su grandi dataset

8.2.2 Limitazioni

- Mancanza di conoscenza del dominio
- Difficoltà nell'interpretazione
- Possibile overfitting
- Costo computazionale elevato

Capitolo 9

Competenze e Strumenti per Feature Engineer

9.1 Percorso Formativo

9.1.1 Competenze Fondamentali

Background Matematico e Statistico

- Statistica descrittiva e inferenziale
- Algebra lineare
- Teoria delle probabilità
- Tecniche di ottimizzazione
- Analisi delle serie temporali

Competenze di Programmazione

- **Python**: linguaggio predominante nel machine learning
- **R**: specializzato in analisi statistica
- **SQL**: gestione database relazionali
- **Scala/Java**: per big data e sistemi distribuiti

Conoscenza del Dominio

- Comprensione dei processi aziendali
- Conoscenza dei KPI rilevanti
- Familiarità con le sfide specifiche del settore
- Capacità di tradurre business requirements in feature

9.2 Librerie e Strumenti Essenziali

9.2.1 Ecosistema Python

Manipolazione Dati

- **Pandas**: manipolazione dataframe
- **NumPy**: calcoli numerici
- **Dask**: computazione parallela per grandi dataset

Machine Learning

- **Scikit-learn**: algoritmi ML e preprocessing
- **XGBoost**: gradient boosting
- **LightGBM**: gradient boosting efficiente
- **CatBoost**: gestione automatica feature categoriche

Feature Engineering Specializzato

- **Feature-engine**: preprocessing avanzato
- **Category-encoders**: encoding categoriche
- **Featuretools**: automated feature engineering
- **TSFresh**: feature per serie temporali

Visualizzazione

- **Matplotlib**: plotting base
- **Seaborn**: visualizzazioni statistiche
- **Plotly**: grafici interattivi
- **Altair**: grammar of graphics

9.2.2 Ecosistema R

- **Tidyverse**: manipolazione dati elegante
- **Caret**: classification and regression training
- **recipes**: preprocessing workflow
- **mlr3**: framework ML moderno

9.3 Ambiente di Sviluppo

9.3.1 IDE e Notebook

- **Jupyter Notebook/Lab**: prototipazione interattiva
- **VS Code**: sviluppo codice
- **PyCharm**: IDE Python completo
- **RStudio**: ambiente R

9.3.2 Controllo Versione

- **Git**: versioning codice
- **DVC**: versioning dati e modelli
- **MLflow**: tracking esperimenti

9.3.3 Deployment e Produzione

- **Docker**: containerizzazione
- **Kubernetes**: orchestrazione
- **Apache Airflow**: pipeline automatizzate
- **Feature stores**: gestione feature in produzione

Capitolo 10

Ruolo del Feature Engineer nel Team

10.1 Responsabilità e Collaborazioni

10.1.1 Collaborazione con Data Engineer

- Definizione requisiti di qualità dei dati
- Progettazione pipeline ETL per feature
- Ottimizzazione performance query
- Gestione data lineage

10.1.2 Collaborazione con Data Scientist

- Comprensione requisiti modelli
- Validazione feature create
- Analisi feature importance
- Ottimizzazione performance modelli

10.1.3 Collaborazione con Business Stakeholder

- Traduzione business requirements
- Validazione logica di business
- Comunicazione insight dalle feature
- Alignment con obiettivi aziendali

10.2 Attività Quotidiane

10.2.1 Sviluppo e Manutenzione

- Sviluppo pipeline feature engineering

- Monitoraggio qualità feature
- Ottimizzazione performance
- Debugging e risoluzione problemi

10.2.2 Ricerca e Sperimentazione

- Esplorazione nuove tecniche
- A/B testing feature
- Analisi impatto business
- Documentazione best practices

10.2.3 Governance e Qualità

- Versioning feature
- Testing automatizzato
- Validazione distribuzione dati
- Monitoraggio drift

Capitolo 11

Casi Studio Pratici

11.1 Caso Studio 1: E-commerce - Previsione Abbandono Carrello

11.1.1 Contesto del Problema

Un'azienda e-commerce vuole prevedere quando un utente abbandonerà il carrello per implementare strategie di retention mirate.

11.1.2 Dataset Originale

- User ID
- Session timestamp
- Product views
- Products added to cart
- Time spent on page
- Device type
- Geographic location

11.1.3 Feature Engineering Implementate

Feature Temporali

```
# Estrazione componenti temporali
df['hour'] = df['timestamp'].dt.hour
df['day_of_week'] = df['timestamp'].dt.dayofweek
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)
df['is_evening'] = (df['hour'] >= 18).astype(int)

# Feature ciclici
df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
```

Feature Comportamentali

```
# Aggregazioni per sessione
df['avg_time_per_product'] = df['time_spent'] / df['
    products_viewed']
df['cart_conversion_rate'] = df['products_in_cart'] / df['
    products_viewed']
df['browse_intensity'] = df['products_viewed'] / df['
    session_duration']

# Feature storiche utente (ultimi 30 giorni)
user_history = df.groupby('user_id').agg({
    'products_viewed': ['mean', 'std', 'max'],
    'products_in_cart': ['mean', 'sum'],
    'session_duration': ['mean', 'std']
}).round(2)
```

Feature Geografiche e Device

```
# Encoding device type
device_encoded = pd.get_dummies(df['device_type'], prefix='
    device')

# Feature geografiche aggregate
location_stats = df.groupby('location')['conversion_rate'].agg([
    'mean', 'count'
]).rename(columns={'mean': 'location_avg_conversion',
                   'count': 'location_frequency'})
```

11.1.4 Risultati

- Miglioramento AUC da 0.72 a 0.89
- Riduzione 35% abbandoni carrello
- ROI campagne retention +150%

11.2 Caso Studio 2: Fintech - Credit Scoring

11.2.1 Contesto del Problema

Una fintech deve valutare il rischio creditizio di nuovi clienti per decisioni di prestito automatizzate.

11.2.2 Feature Engineering per Dati Finanziari

Ratios Finanziari

```
# Ratios di liquidita
df['debt_to_income'] = df['total_debt'] / df['monthly_income']
df['expense_ratio'] = df['monthly_expenses'] / df['
    monthly_income']
df['savings_rate'] = (df['monthly_income'] - df['
    monthly_expenses']) / df['monthly_income']

# Indicatori di stabilita
df['employment_stability'] = df['months_current_job'] / df['
    total_work_experience']
df['address_stability'] = df['months_current_address'] / df['age
    ']
```

Feature Temporali sui Pagamenti

```
# Analisi storico pagamenti
payment_features = payment_history.groupby('customer_id').agg({
    'days_late': ['mean', 'max', 'std', 'count'],
    'payment_amount': ['mean', 'std', 'sum'],
    'is_late': ['sum', 'mean']  # numero e percentuale pagamenti
    in ritardo
})

# Trend recenti (ultimi 6 mesi)
recent_payments = payment_history[
    payment_history['date'] >= (datetime.now() - timedelta(days
    =180))
]

trend_features = recent_payments.groupby('customer_id').agg({
    'days_late': lambda x: np.polyfit(range(len(x)), x, 1)[0]  #
    trend lineare
})
```

Feature di Segmentazione

```
# Segmentazione demografica
df['age_group'] = pd.cut(df['age'], bins=[18, 25, 35, 50, 100],
                         labels=['Young', 'Adult', 'Middle', '
    Senior'])

df['income_quartile'] = pd.qcut(df['monthly_income'], q=4,
                                labels=['Low', 'Medium-Low', '
    Medium-High', 'High'])

# Feature composite
df['risk_score'] = (
    df['debt_to_income'] * 0.4 +
    df['late_payment_rate'] * 0.3 +
    (1 - df['employment_stability']) * 0.2 +
    df['credit_utilization'] * 0.1
)
```

11.2.3 Risultati

- Precisione modello: 94%
- Riduzione default rate: 28%
- Tempo decisione: da 48h a 5 minuti
- Incremento approvazioni clienti validi: 15%

11.3 Caso Studio 3: Healthcare - Predizione Readmission

11.3.1 Feature Engineering per Dati Medici

Feature Cliniche Composite

```
# Indici di comorbidita
df['charlson_score'] = (
    df['diabetes'] * 1 +
    df['heart_disease'] * 1 +
    df['copd'] * 1 +
    df['renal_disease'] * 2 +
    df['cancer'] * 2
)

# Stabilita vitali
```

```
vital_signs = ['blood_pressure', 'heart_rate', 'temperature', 'oxygen_sat']
for vital in vital_signs:
    df[f'{vital}_stability'] = 1 / (df[f'{vital}_std'] + 1)
```

Feature Temporali Ospedaliere

```
# Pattern di utilizzo servizi
df['admission_frequency'] = df.groupby('patient_id')['admission_date'].transform('count')
df['days_since_last_admission'] = (
    df['current_admission'] - df.groupby('patient_id')['admission_date'].shift(1)
).dt.days

# Seasonal patterns
df['admission_month'] = df['admission_date'].dt.month
df['flu_season'] = df['admission_month'].isin([11, 12, 1, 2, 3]).astype(int)
```

Capitolo 12

Best Practices e Considerazioni Avanzate

12.1 Pipeline di Feature Engineering

12.1.1 Organizzazione del Codice

```python
class FeatureEngineeringPipeline:
    def __init__(self, config):
        self.config = config
        self.transformers = {}

    def add_transformer(self, name, transformer):
        self.transformers[name] = transformer

    def fit(self, X, y=None):
        for name, transformer in self.transformers.items():
            if hasattr(transformer, 'fit'):
                transformer.fit(X, y)
        return self

    def transform(self, X):
        X_transformed = X.copy()
        for name, transformer in self.transformers.items():
            X_transformed = transformer.transform(X_transformed)
        return X_transformed

    def fit_transform(self, X, y=None):
        return self.fit(X, y).transform(X)

# Utilizzo
pipeline = FeatureEngineeringPipeline(config)
pipeline.add_transformer('scaler', StandardScaler())
pipeline.add_transformer('encoder', OneHotEncoder(drop='first'))
pipeline.add_transformer('selector', SelectKBest(k=20))

```

```
X_processed = pipeline.fit_transform(X_train, y_train)
```

Listing 12.1: Struttura pipeline modulare

12.1.2 Gestione della Data Leakage

Temporal Leakage

```
# ERRATO: utilizza informazioni future
df['future_sales'] = df['sales'].shift(-1)  #

# CORRETTO: utilizza solo informazioni passate
df['sales_lag_1'] = df['sales'].shift(1)    #
df['sales_ma_7'] = df['sales'].rolling(7).mean()  #
```

Target Leakage

```
# ERRATO: feature altamente correlata al target
df['purchase_amount_next_month'] = df['target']  #

# CORRETTO: feature indipendente dal target
df['avg_purchase_last_3_months'] = (
    df['purchase_amount'].rolling(3).mean().shift(1)
)  #
```

12.2 Validazione e Testing

12.2.1 Unit Testing per Feature Engineering

```
import unittest
import pandas as pd
import numpy as np

class TestFeatureEngineering(unittest.TestCase):

    def setUp(self):
        self.sample_data = pd.DataFrame({
            'date': pd.date_range('2023-01-01', periods=100),
            'sales': np.random.randint(100, 1000, 100),
            'category': np.random.choice(['A', 'B', 'C'], 100)
        })

    def test_temporal_features(self):
```

```
        result = create_temporal_features(self.sample_data, '
    date')

        # Verifica presenza colonne attese
        expected_cols = ['year', 'month', 'day_of_week', '
    is_weekend']
        for col in expected_cols:
            self.assertIn(col, result.columns)

        # Verifica range valori
        self.assertTrue(result['month'].between(1, 12).all())
        self.assertTrue(result['day_of_week'].between(0, 6).all
    ())
        self.assertTrue(result['is_weekend'].isin([0, 1]).all())

    def test_rolling_features(self):
        result = create_rolling_features(self.sample_data, '
    sales', [7, 30])

        # Verifica nessun valore futuro utilizzato
        self.assertTrue(result['sales_ma_7'].iloc[:6].isna().all
    ())

        # Verifica calcolo corretto
        expected_ma = self.sample_data['sales'].iloc[0:7].mean()
        self.assertAlmostEqual(result['sales_ma_7'].iloc[6],
    expected_ma)

if __name__ == '__main__':
    unittest.main()
```

Listing 12.2: Test per funzioni di feature engineering

12.2.2 Cross-Validation Consapevole del Tempo

```
from sklearn.model_selection import TimeSeriesSplit

def time_aware_validation(X, y, model, n_splits=5):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    scores = []

    for train_idx, val_idx in tscv.split(X):
        X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
        y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

        # Feature engineering solo su training set
        fe_pipeline = create_feature_pipeline()
        fe_pipeline.fit(X_train, y_train)
```

```
14
15          X_train_fe = fe_pipeline.transform(X_train)
16          X_val_fe = fe_pipeline.transform(X_val)
17
18          # Training e validazione
19          model.fit(X_train_fe, y_train)
20          score = model.score(X_val_fe, y_val)
21          scores.append(score)
22
23      return np.mean(scores), np.std(scores)
```

Listing 12.3: Time Series Cross-Validation

12.3 Monitoraggio in Produzione

12.3.1 Feature Drift Detection

```
1  from scipy import stats
2  import warnings
3
4  class FeatureDriftMonitor:
5      def __init__(self, reference_data, threshold=0.05):
6          self.reference_data = reference_data
7          self.threshold = threshold
8          self.reference_stats = self._compute_stats(
       reference_data)
9
10     def _compute_stats(self, data):
11         stats_dict = {}
12         for col in data.select_dtypes(include=[np.number]).
       columns:
13             stats_dict[col] = {
14                 'mean': data[col].mean(),
15                 'std': data[col].std(),
16                 'distribution': data[col].values
17             }
18         return stats_dict
19
20     def detect_drift(self, new_data):
21         drift_detected = {}
22
23         for col in self.reference_stats.keys():
24             if col in new_data.columns:
25                 # Kolmogorov-Smirnov test
26                 ks_stat, p_value = stats.ks_2samp(
27                     self.reference_stats[col]['distribution'],
28                     new_data[col].values
```

```
                )

                drift_detected[col] = {
                    'drift': p_value < self.threshold,
                    'p_value': p_value,
                    'ks_statistic': ks_stat
                }

                if drift_detected[col]['drift']:
                    warnings.warn(f"Drift detected in feature {col}: p={p_value:.4f}")

        return drift_detected

# Utilizzo
monitor = FeatureDriftMonitor(X_train)
drift_results = monitor.detect_drift(X_production)
```

Listing 12.4: Monitoraggio drift delle feature

12.3.2 Performance Monitoring

```
import mlflow
import time

class FeaturePerformanceTracker:
    def __init__(self, experiment_name):
        mlflow.set_experiment(experiment_name)

    def track_feature_creation(self, feature_name, creation_func, data):
        with mlflow.start_run():
            start_time = time.time()

            try:
                result = creation_func(data)
                execution_time = time.time() - start_time

                # Log metriche
                mlflow.log_metric("execution_time", execution_time)
                mlflow.log_metric("feature_count", len(result.columns))
                mlflow.log_metric("memory_usage_mb",
                                  result.memory_usage(deep=True).sum() / 1024**2)

                # Log proprieta statistiche
```

```
                for col in result.select_dtypes(include=[np.
    number]).columns:
                    mlflow.log_metric(f"{col}_mean", result[col
    ].mean())
                    mlflow.log_metric(f"{col}_std", result[col].
    std())
                    mlflow.log_metric(f"{col}_null_pct",
                                      result[col].isnull().mean()
    * 100)

                mlflow.log_param("feature_name", feature_name)
                mlflow.log_param("input_shape", data.shape)
                mlflow.log_param("output_shape", result.shape)

                return result

            except Exception as e:
                mlflow.log_param("error", str(e))
                raise

tracker = FeaturePerformanceTracker("
    feature_engineering_monitoring")
```

Listing 12.5: Monitoraggio performance feature

Capitolo 13

Tendenze Future e Tecnologie Emergenti

13.1 Deep Learning e Feature Learning

13.1.1 Automated Feature Extraction

Il deep learning sta rivoluzionando il feature engineering attraverso l'apprendimento automatico di rappresentazioni:

Representation Learning

- **Autoencoders**: apprendimento di encoding compatti
- **Variational Autoencoders (VAE)**: generazione di feature probabilistiche
- **Generative Adversarial Networks (GAN)**: creazione feature sintetiche

Transfer Learning per Feature

```
from transformers import AutoModel, AutoTokenizer
import torch

class TransformerFeatureExtractor:
    def __init__(self, model_name='bert-base-uncased'):
        self.tokenizer = AutoTokenizer.from_pretrained(
   model_name)
        self.model = AutoModel.from_pretrained(model_name)
        self.model.eval()

    def extract_features(self, texts):
        features = []

        for text in texts:
            inputs = self.tokenizer(text, return_tensors='pt',
                                    truncation=True, padding=True)

```

```
            with torch.no_grad():
                outputs = self.model(**inputs)
                # Utilizza rappresentazione [CLS] token
                feature = outputs.last_hidden_state[:, 0, :].
    numpy()
                features.append(feature.flatten())

        return np.array(features)

# Utilizzo per feature testuali
extractor = TransformerFeatureExtractor()
text_features = extractor.extract_features(df['
    product_description'].tolist())
```

Listing 13.1: Transfer learning per feature extraction

13.2 Quantum Feature Engineering

13.2.1 Quantum Computing per Ottimizzazione

Le tecnologie quantistiche promettono di rivoluzionare l'ottimizzazione delle feature:

- **Quantum Annealing**: ottimizzazione combinatoriale per feature selection
- **Quantum ML**: algoritmi quantistici per pattern recognition
- **Quantum Embeddings**: rappresentazioni in spazi di Hilbert

13.3 Edge Computing e Real-time Features

13.3.1 Feature Engineering Distribuita

```
from kafka import KafkaProducer, KafkaConsumer
import json
import pandas as pd

class RealTimeFeatureProcessor:
    def __init__(self, input_topic, output_topic):
        self.consumer = KafkaConsumer(
            input_topic,
            bootstrap_servers=['localhost:9092'],
            value_deserializer=lambda m: json.loads(m.decode('
    utf-8'))
        )
        self.producer = KafkaProducer(
            bootstrap_servers=['localhost:9092'],
            value_serializer=lambda v: json.dumps(v).encode('utf
    -8')
```

```
        )
        self.output_topic = output_topic

    def process_stream(self):
        for message in self.consumer:
            raw_data = message.value

            # Feature engineering real-time
            features = self.create_real_time_features(raw_data)

            # Invio feature processate
            self.producer.send(self.output_topic, features)

    def create_real_time_features(self, data):
        # Implementazione feature engineering
        features = {
            'user_id': data['user_id'],
            'timestamp': data['timestamp'],
            'session_duration': data['current_time'] - data['session_start'],
            'page_views_per_minute': data['page_views'] / (data['session_duration'] / 60),
            'is_mobile': data['device_type'] == 'mobile'
        }
        return features

processor = RealTimeFeatureProcessor('raw_events', 'processed_features')
processor.process_stream()
```

Listing 13.2: Feature engineering real-time con Apache Kafka

13.4 Explainable AI e Feature Interpretation

13.4.1 SHAP Values per Feature Analysis

```
import shap
import matplotlib.pyplot as plt

class FeatureExplainer:
    def __init__(self, model, X_background):
        self.model = model
        self.explainer = shap.Explainer(model, X_background)

    def explain_features(self, X_test, feature_names):
        shap_values = self.explainer(X_test)

```

```
        # Summary plot
        shap.summary_plot(shap_values, X_test, feature_names=
    feature_names)

        # Feature importance
        feature_importance = pd.DataFrame({
            'feature': feature_names,
            'importance': np.abs(shap_values.values).mean(0)
        }).sort_values('importance', ascending=False)

        return feature_importance, shap_values

    def analyze_feature_interactions(self, X_test, feature_names
    ):
        # Analisi interazioni tra feature
        shap_interaction_values = self.explainer.
    shap_interaction_values(X_test)

        # Heatmap interazioni
        interaction_matrix = np.abs(shap_interaction_values).
    mean(0)

        plt.figure(figsize=(12, 10))
        sns.heatmap(interaction_matrix,
                    xticklabels=feature_names,
                    yticklabels=feature_names,
                    annot=True, cmap='viridis')
        plt.title('Feature Interaction Strength')
        plt.show()

        return interaction_matrix

explainer = FeatureExplainer(trained_model, X_train)
importance, shap_values = explainer.explain_features(X_test,
    feature_names)
```

Listing 13.3: Analisi feature con SHAP

Capitolo 14

Conclusioni

14.1 Riepilogo dei Concetti Chiave

Il feature engineering rappresenta una disciplina fondamentale nel machine learning e nell'analisi dei dati che richiede una combinazione unica di:

- **Competenze tecniche**: padronanza di algoritmi, statistica e programmazione
- **Creatività**: capacità di immaginare nuove rappresentazioni dei dati
- **Conoscenza del dominio**: comprensione profonda del contesto aziendale
- **Pensiero critico**: valutazione dell'impatto e della rilevanza delle feature

14.1.1 Fattori Critici di Successo

1. **Comprensione del Problema**: prima di creare feature, è essenziale comprendere il problema aziendale e i suoi vincoli
2. **Qualità dei Dati**: feature eccellenti non possono compensare dati di bassa qualità
3. **Iterazione e Sperimentazione**: il feature engineering è un processo iterativo che richiede continua sperimentazione
4. **Validazione Rigorosa**: ogni feature deve essere validata per evitare overfitting e data leakage
5. **Scalabilità**: le soluzioni devono essere progettate per scalare in produzione

14.2 Impatto sul Business

Il feature engineering di qualità può determinare:

- **Miglioramento delle Prestazioni**: incrementi significativi nell'accuratezza dei modelli (spesso 10-30%)
- **Riduzione dei Costi**: modelli più efficienti richiedono meno risorse computazionali
- **Time-to-Market**: feature ben progettate accelerano lo sviluppo di soluzioni ML

- **Interpretabilità**: feature meaningful facilitano l'adozione aziendale
- **ROI Superiore**: migliori prestazioni si traducono in maggiore valore di business

14.3 Raccomandazioni per il Futuro

14.3.1 Per i Professionisti

- **Formazione Continua**: rimanere aggiornati su nuove tecniche e strumenti
- **Portfolio Progetti**: sviluppare un portfolio che dimostri competenze diverse
- **Specializzazione Verticale**: approfondire la conoscenza in settori specifici
- **Collaborazione**: sviluppare soft skills per lavorare efficacemente in team multidisciplinari

14.3.2 Per le Organizzazioni

- **Investimento in Talenti**: attrarre e sviluppare feature engineer qualificati
- **Cultura Data-Driven**: promuovere una cultura aziendale orientata ai dati
- **Infrastruttura**: investire in piattaforme e strumenti per il feature engineering
- **Governance**: stabilire processi e standard per la gestione delle feature

14.4 Tendenze Emergenti

Il futuro del feature engineering sarà caratterizzato da:

- **Maggiore Automazione**: strumenti AutoML sempre più sofisticati
- **Real-time Processing**: feature engineering in tempo reale per applicazioni critiche
- **Explainable AI**: crescente enfasi sull'interpretabilità delle feature
- **Edge Computing**: elaborazione distribuita per applicazioni IoT e mobile
- **Quantum Computing**: nuove possibilità di ottimizzazione quantistica

14.5 Considerazioni Finali

Il feature engineering rimane più un'arte che una scienza, richiedendo intuizione, esperienza e creatività. Tuttavia, con l'approccio metodologico corretto, gli strumenti appropriati e una profonda comprensione del dominio, è possibile creare feature che trasformano dati grezzi in insight di valore.

La capacità di creare feature efficaci rappresenta spesso la differenza tra un progetto di machine learning di successo e uno che fallisce. Investire nelle competenze di feature engineering è quindi fondamentale per qualsiasi organizzazione che voglia sfruttare appieno il potenziale dei propri dati.

Come il fisico Richard Feynman disse una volta: *"What I cannot create, I do not understand"*. Nel contesto del feature engineering, questo si traduce in: ciò che non sappiamo rappresentare nei dati, non possiamo utilizzarlo efficacemente per le nostre previsioni.

Il percorso per diventare un feature engineer esperto richiede dedizione e continuo apprendimento, ma le opportunità professionali in questo campo sono numerose e in crescita costante. Con il giusto mix di formazione teorica, esperienza pratica e comprensione del business, è possibile costruire una carriera gratificante in questo settore dinamico e in rapida evoluzione.

Bibliografia

[1] DataMasters.it, *Cos'è l'ingegneria delle funzionalità*, https://datamasters.it/blog/cose-lingegneria-delle-funzionalita/

[2] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*, O'Reilly Media, 2018.

[3] M. Kuhn and K. Johnson, *Feature Engineering and Selection: A Practical Approach for Predictive Models*, CRC Press, 2019.

[4] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow*, 3rd Edition, Packt Publishing, 2019.

[5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, O'Reilly Media, 2019.

[6] J. Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*, Machine Learning Mastery, 2020.

[7] DataCamp, *Feature Engineering in Machine Learning: A Practical Guide*, https://www.datacamp.com/tutorial/feature-engineering

[8] P. Grabiński, *Feature Engineering for Machine Learning: 10 Examples*, KDnuggets, 2018.

[9] Built In, *Feature Engineering Explained*, https://builtin.com/articles/feature-engineering

[10] IBM, *What is feature engineering?*, IBM Think Topics, 2025.