meta_learning_eda

November 11, 2025

1 Agentic ML Builder

1.0.1 The Agentic ML Builder is more of a ML Scafolding code generator targeting improvement of productivity of ML Engineers

1.0.2 The Agentic ML Builder uses some datasets internally for Meta-Learning to Learn which ML algorithm performs best for given dataset characteristics.

1.0.3 This notebook is used to perform EDA on Meta-Learning with OpenML datasets

1.1 Meta-Learning Dataset EDA

This notebook performs exploratory data analysis on the synthetic meta-learning dataset used in the Agentic ML Builder MVP. It demonstrates variable exploration, missing values handling, feature engineering, and correlation analysis.

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
meta = pd.read_csv('./meta_learning.csv')
meta.head()
```

[1]:
```
  dataset_name  num_instances  num_features  num_numeric_features  \
0         ds_0          15895           294                   195
1         ds_1            960           100                    93
2         ds_2           5490           173                   161
3         ds_3          12064           361                   221
4         ds_4          11384           215                   155

   num_categorical_features  missing_value_ratio       task_type  \
0                        99                0.051  classification
1                         7                0.125      clustering
2                        12                0.014  classification
3                       140                0.010  classification
4                        60                0.109  classification

        algorithm  accuracy  rmse
0    RandomForest     0.694   NaN
1          KMeans       NaN   NaN
```

```
2  LogisticRegression     0.815    NaN
3  LogisticRegression     0.663    NaN
4        RandomForest     0.756    NaN
```

1.2 Variable datatypes and summary

[2]:
```
meta.info()
meta.describe(include='all')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   dataset_name            100 non-null    object
 1   num_instances           100 non-null    int64
 2   num_features            100 non-null    int64
 3   num_numeric_features    100 non-null    int64
 4   num_categorical_features 100 non-null   int64
 5   missing_value_ratio     100 non-null    float64
 6   task_type               100 non-null    object
 7   algorithm               100 non-null    object
 8   accuracy                74 non-null     float64
 9   rmse                    21 non-null     float64
dtypes: float64(3), int64(4), object(3)
memory usage: 7.9+ KB
```

[2]:
```
       dataset_name  num_instances  num_features  num_numeric_features  \
count           100     100.000000    100.000000            100.000000
unique          100            NaN           NaN                   NaN
top            ds_0            NaN           NaN                   NaN
freq              1            NaN           NaN                   NaN
mean            NaN    9222.040000    245.190000            172.590000
std             NaN    5712.819106    136.832397             97.006518
min             NaN     261.000000      6.000000              3.000000
25%             NaN    4308.500000    137.250000             95.750000
50%             NaN    8650.000000    233.000000            160.500000
75%             NaN   13659.250000    360.000000            239.000000
max             NaN   19869.000000    486.000000            397.000000

        num_categorical_features  missing_value_ratio       task_type  \
count                 100.000000           100.000000             100
unique                       NaN                  NaN               3
top                          NaN                  NaN  classification
freq                         NaN                  NaN              74
mean                   72.600000             0.054620             NaN
std                    56.636177             0.051859             NaN
```

```
min               2.000000          0.001000          NaN
25%              23.000000          0.017500          NaN
50%              64.000000          0.039500          NaN
75%              99.000000          0.080500          NaN
max             236.000000          0.267000          NaN

           algorithm   accuracy       rmse
count            100  74.000000  21.000000
unique             6        NaN        NaN
top     RandomForest        NaN        NaN
freq              54        NaN        NaN
mean             NaN   0.739946  33.915095
std              NaN   0.066245  11.217783
min              NaN   0.579000  14.238000
25%              NaN   0.691750  26.561000
50%              NaN   0.741500  34.534000
75%              NaN   0.800000  40.860000
max              NaN   0.845000  51.414000
```

1.3 Missing values and basic cleaning

[3]:
```python
print(meta.isnull().sum())
# For demonstration, fill missing numeric with median
meta_clean = meta.copy()
meta_clean['accuracy'] = meta_clean['accuracy'].fillna(meta_clean['accuracy'].
 ↪median())
meta_clean['rmse'] = meta_clean['rmse'].fillna(meta_clean['rmse'].median())
meta_clean.isnull().sum()
```

```
dataset_name                 0
num_instances                0
num_features                 0
num_numeric_features         0
num_categorical_features     0
missing_value_ratio          0
task_type                    0
algorithm                    0
accuracy                    26
rmse                        79
dtype: int64
```

[3]:
```
dataset_name                0
num_instances               0
num_features                0
num_numeric_features        0
num_categorical_features    0
missing_value_ratio         0
```

```
task_type                 0
algorithm                 0
accuracy                  0
rmse                      0
dtype: int64
```

1.4 Feature engineering

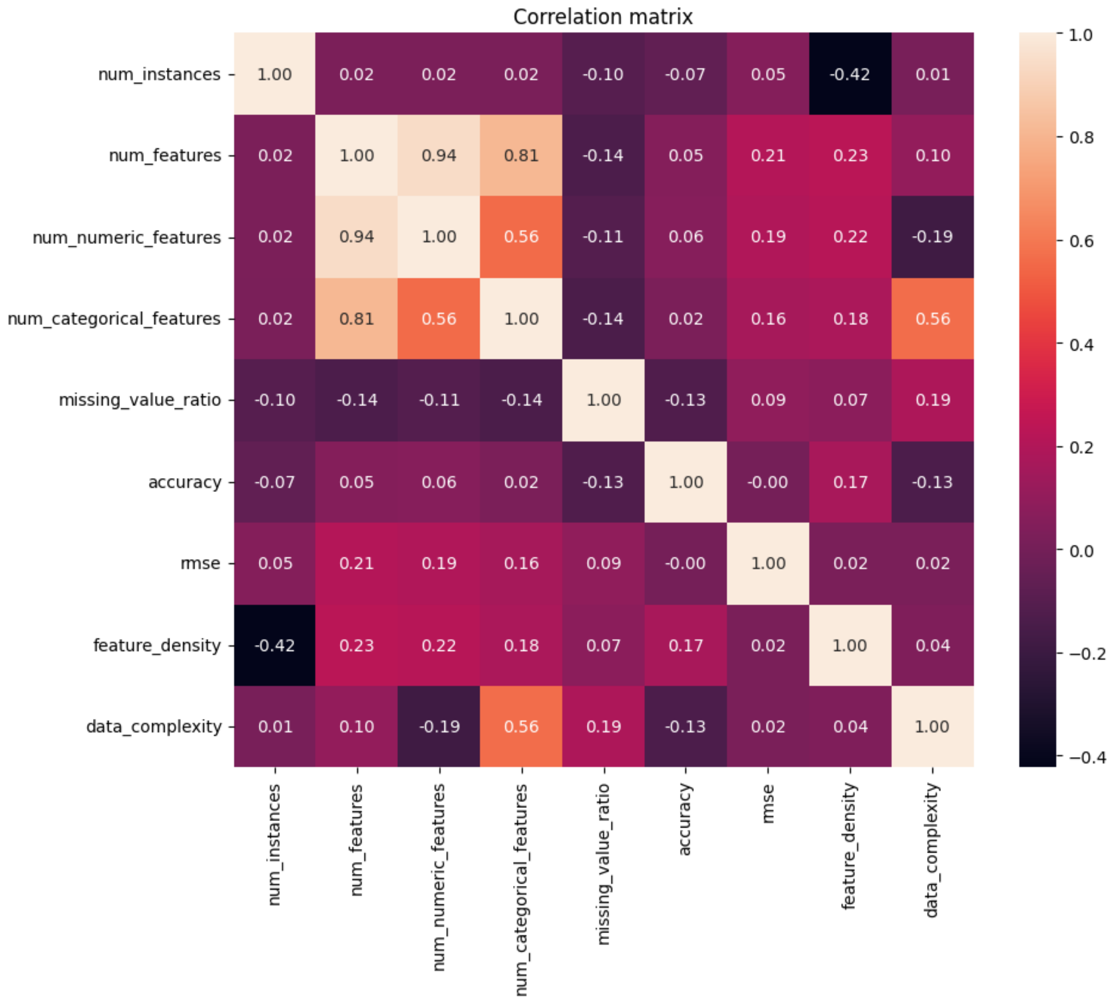
[4]:
```
meta_clean['feature_density'] = meta_clean['num_features'] /␣
  ↪meta_clean['num_instances']
meta_clean['data_complexity'] = (meta_clean['missing_value_ratio']*0.4) +␣
  ↪(meta_clean['num_categorical_features']/meta_clean['num_features']*0.6)
meta_clean[['feature_density','data_complexity']].describe()
```

[4]:

	feature_density	data_complexity
count	100.000000	100.000000
mean	0.073156	0.193542
std	0.179368	0.078698
min	0.000840	0.047218
25%	0.015230	0.133317
50%	0.026149	0.187001
75%	0.047866	0.259349
max	1.318339	0.339600

1.5 Correlation heatmap (numeric features)

[5]:
```
corr = meta_clean.select_dtypes(include=['number']).corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, fmt='.2f')
plt.title('Correlation matrix')
plt.show()
```



Each cell in the heatmap represents the **Pearson correlation coefficient** between two variables.

Values range from -1 to +1:

+1 → perfect positive correlation (as one increases, the other also increases).

-1 → perfect negative correlation (as one increases, the other decreases).

0 → no linear relationship.

The color intensity encodes the strength of correlation: lighter shades represent strong positive correlations, darker shades near zero or black represent weak or negative correlations.

1.5.1 Correlation Matrix Insights

- The correlation matrix reveals relationships among dataset characteristics and performance metrics.
- **Strong positive correlations** are observed between:

- `num_features` `num_numeric_features` (0.94)

- `num_features` `num_categorical_features` (0.81)

- `num_categorical_features` `data_complexity` (0.56)
  → Indicating that datasets with more features generally have both more numeric and categorical attributes, contributing to higher data complexity.

- **Negative correlation:** `num_instances` `feature_density` (-0.42), suggesting larger datasets tend to be less dense in feature space.
- **Performance metrics** (`accuracy`, `mse`) show weak correlations with dataset attributes, implying that model performance depends on more complex factors than just dataset size or feature count.
- Overall, dataset structural attributes are interrelated, but they have limited direct influence on predictive performance.

[ ]:

```

```