04_transforms

June 21, 2025

1 Feature Engineering

- Feature engineering is to transform the data in such a way that the information content is easily exposed to the model.
- This statement can mean many things and highly depends on what exactly is "the model".0
- As we have seen, we are using many tools in combination to manipulate data. Thus far, we have encountered pandas, Dask, and sklearn in this course, but there are many more (PySpark, SQL, DAX, M, R, etc.)
- It is important to discuss which tools are the right ones, specifically in the context of data leakage.

1.1 Transform using pandas/Dask/SQL or sklearn?

- Depending on the perspective, the answer could be neither, pandas, or sklearn:
  - Neither:
    - Most join and filtering should be done closer to the source using a database or parquet/Dask operation.
    - Map-Reduce and Group-by-Aggregate ("data warehousing") operations.
    - Indexing and reshuffling.
  - Pandas, Dask, or PySpark:
    - Renames tasks.
    - Use python libraries like pandas, Dask, or pySpark to add contemporaneous feature, time-series manipulation (for example, adding lags), parallel computation (using Dask or pySpark).
    - Do not use these libraries for sample-dependent features.
  - Use sklearn, pytorch:
    - Use python libraries like sklearn or pytorch to add features that are sample-dependent like scaling and normalization, one-hot encoding, tokenization, and vectorization.
    - Model-depdenent transformations: PCA, embeddings, iterative/knn imputation, etc.
- Decisions must be guided by optimization criteria (time and resources) while avoiding data leakage.

1.2 Example Transforms in sklearn

The list below is found in Scikit's Documentation, which also includes convenience interfaces for the classes below.

Work with categorical variables:

- `preprocessing.Binarizer(*[, threshold, copy])`: Binarize data (set feature values to 0 or 1) according to a threshold.
- `preprocessing.KBinsDiscretizer([n_bins, ...])`: Bin continuous data into intervals.
- `preprocessing.LabelBinarizer(*[, neg_label, ...])`: Binarize labels in a one-vs-all fashion.
- `preprocessing.LabelEncoder()`: Encode target labels with value between 0 and n_classes-1.
- `preprocessing.MultiLabelBinarizer(*[, ...])`: Transform between iterable of iterables and a multilabel format.
- `preprocessing.OneHotEncoder(*[, categories, ...])`: Encode categorical features as a one-hot numeric array.
- `preprocessing.OrdinalEncoder(*[, ...])`: Encode categorical features as an integer array.

Scale and normalize:

- `preprocessing.StandardScaler(*[, copy, ...])`: Standardize features by removing the mean and scaling to unit variance.
- `preprocessing.MaxAbsScaler(*[, copy])`: Scale each feature by its maximum absolute value.
- `preprocessing.MinMaxScaler([feature_range, ...])`: Transform features by scaling each feature to a given range.
- `preprocessing.Normalizer([norm, copy])`: Normalize samples individually to unit norm.
- `preprocessing.RobustScaler(*[, ...])`: Scale features using statistics that are robust to outliers.

Nonlinear transforms:

- `preprocessing.FunctionTransformer([func, ...])`: Constructs a transformer from an arbitrary callable.
- `preprocessing.KernelCenterer()`: Center an arbitrary kernel matrix
- `preprocessing.PolynomialFeatures([degree, ...])`: Generate polynomial and interaction features.
- `preprocessing.PowerTransformer([method, ...])`: Apply a power transform featurewise to make data more Gaussian-like.
- `preprocessing.QuantileTransformer(*[, ...])`: Transform features using quantiles information.
- `preprocessing.SplineTransformer([n_knots, ...])`: Generate univariate B-spline bases for features.
- `preprocessing.TargetEncoder([categories, ...])`: Target Encoder for regression and classification targets.

1.3 What are we doing?

1.3.1 The Objectives

Build a pipeline that:

- Add indicators:

  - SME indicated that a Debt-to-Ratio > 100% is too high.
  - Missing values indicator for `monthly_income` and `num_dependents`.

- Impute missing values, where required.
- Standardize variables.
- Evaluate if a transform (Yeo-Johnson or Box-Cox) of selected variables (debt_ratio, monthly_income, and revolving_unsecured_line_utilization) is beneficial.

Feature selection:

- We are looking for informative features: their contribution to prediction is valuable.
- We prefer parsimonious models.
- We want to retain evidence of our work and afford reproducibility.

2 Data Source

- For this example, we will use Give Me Some Credit from Kaggle, a widely refered example.
- To run the examples below, download the data set and extract cs-training.csv to `../05_src/data/credit/`.

2.1 Our data

[ ]:
```
# Load environment variables
%load_ext dotenv
%dotenv
# Add src to path
import os
import sys
sys.path.append(os.getenv('SRC_DIR'))

# Standard libraries
import pandas as pd
import numpy as np


# Load data
ft_file = os.getenv("CREDIT_DATA")
df_raw = pd.read_csv(ft_file)
```

[ ]:
```
df_raw.info()
```

[ ]:
```
df = df_raw.drop(columns = ["Unnamed: 0"]).rename(
    columns = {
        'SeriousDlqin2yrs': 'delinquency',
        'RevolvingUtilizationOfUnsecuredLines':␣
 ↪'revolving_unsecured_line_utilization',
        'age': 'age',
        'NumberOfTime30-59DaysPastDueNotWorse': 'num_30_59_days_late',
```

```
        'DebtRatio': 'debt_ratio',
        'MonthlyIncome': 'monthly_income',
        'NumberOfOpenCreditLinesAndLoans': 'num_open_credit_loans',
        'NumberOfTimes90DaysLate':  'num_90_days_late',
        'NumberRealEstateLoansOrLines': 'num_real_estate_loans',
        'NumberOfTime60-89DaysPastDueNotWorse': 'num_60_89_days_late',
        'NumberOfDependents': 'num_dependents'
    }
).assign(
    high_debt_ratio = lambda x: (x['debt_ratio'] > 1)*1,
    missing_monthly_income = lambda x: x['monthly_income'].isna()*1,
    missing_num_dependents = lambda x: x['num_dependents'].isna()*1,
)
```

2.2 Manual Solution

- To get deeper insights into the task, first approach it manually.

[ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, PowerTransformer
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.naive_bayes import GaussianNB

num_cols = ['revolving_unsecured_line_utilization', 'age',
       'num_30_59_days_late', 'debt_ratio', 'monthly_income',
       'num_open_credit_loans', 'num_90_days_late', 'num_real_estate_loans',
       'num_60_89_days_late', 'num_dependents',
       # Although expressed as numbers, these columns are boolean:
       # 'high_debt_ratio',
       # 'missing_monthly_income',
       # 'missing_num_dependents'
       ]

pipe_num_simple = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median')),
    ('standardizer', StandardScaler())
])

ctransform_simple= ColumnTransformer([
    ('numeric_simple', pipe_num_simple, num_cols),
], remainder='passthrough')

pipe_simple = Pipeline([
    ('preprocess', ctransform_simple),
    ('model', GaussianNB())
```

```
])
pipe_simple
```

2.3 Cross-validation of simple pipeline

[ ]:
```
X = df.drop(columns = 'delinquency')
Y = df['delinquency']

scoring = ['neg_log_loss', 'roc_auc', 'f1', 'accuracy', 'precision', 'recall']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 42)
```

[ ]:
```
res_simple_dict = cross_validate(pipe_simple, X_train, Y_train, cv = 5, scoring␣
 ↪= scoring)
res_simple = pd.DataFrame(res_simple_dict).assign(experiment = 1)
res_simple
```

On average, we obtain a log-loss of about 0.362.

[ ]:
```
res_simple.mean()
```

2.4 Alternative Pipeline

- The pipeline below is more complex:
  - Treat selected numericals using Yeo-Johnson transformation.
  - Treat other numericals with scaling only.
  - Do not treat booleans.

[ ]:
```
num_cols = ['age',
       'num_30_59_days_late', 'num_open_credit_loans', 'num_90_days_late',␣
 ↪'num_real_estate_loans',
       'num_60_89_days_late', 'num_dependents',
       ]

num_cols_transform = ['revolving_unsecured_line_utilization', 'debt_ratio',␣
 ↪'monthly_income',]

pipe_num_simple = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median')),
    ('standardizer', StandardScaler())
])

pipe_num_yj = Pipeline([
    ('imputer', SimpleImputer(strategy = 'median')),
    ('standardizer', StandardScaler()),
```

```
    ('transform', PowerTransformer(method='yeo-johnson'))
])

ctramsform_yj = ColumnTransformer([
    ('numeric_std', pipe_num_simple, num_cols),
    ('numeric_yj', pipe_num_yj, num_cols_transform),
], remainder='passthrough')

pipe_yj = Pipeline([
    ('preprocess', ctramsform_yj),
    ('clf', GaussianNB())
])
pipe_yj
```

[ ]:
```
res_yj_dict = cross_validate(pipe_yj, X_train, Y_train, cv = 5, scoring =␣
↪scoring)
res_yj = pd.DataFrame(res_yj_dict).assign(experiment = 2)
res_yj
```

We obtained a greater loss of 0.443, therefore the additional feature is not profitable.

[ ]:
```
res_yj.mean()
```

3 Reflection

- We are currently evaluating two feature engineering procedures using the same classifier.
  - However, feature engineering is classifier-dependent: each classifier is a specialized tool to learn a certain type of hypothesis.
  - Different classifiers will benefit from different type of engineered features (see, for example, Khun and Silge's recommendations on TMWR.org).
- We are producing data from our experiments.
  - The data that we produced is more or less structured: we are using standard performance metrics, for instance.
  - Each preprocessing pipeline will be different and may accept different configuration parameters.
  - Likewise, classifiers will tend to have different configuration parameters.
- We modify code to produce experiments:
  - Our experiment results will be a function of our algorithm's logic, its implementation (code), and our data.
  - Code tracking is doen with Git.
  - Data tracking is in development.

It is generally a good idea to use software for experiment tracking once you move out of the Proof of Concept stage. Some solutions include:

- ML Flow.
- Weights & Balances.
- Sacred.

4 MLFlow

- MLFlow is a software tool that automates taks related to experiment tracking:
  - Keep track of experiment parameters.
  - Save configuration+s for individual experiment runs in files or databases.
  - Store models and other artifacts to an object store.
- A few features that may be useful:
  - Keep track of code and artifacts associated with experiment.
  - Store experiment run times and system characteristics.
  - Work with different backend stores ("Observers").

4.1 Our First Experiment

Continuing with our example, the following setup will track an experiment to measure the performance of a model pipeline. The main file for this experiment is `./05_src/credit/exp__logistic_simple.py`. You can run this experiment from the `05_src/` folder using `python -m credit.exp__logistic_simple`.

After running the experiment, take a look at MLFlow by navigating to `http://localhost:5001`.