Satej Soman
CAPP30254: Machine Learning for Public Policy
Spring 2019

HW 2
MACHINE LEARNING PIPELINE

Contents

1 Pipeline Library Design **1**
1.1 Overview . . . 1
1.2 Design Decisions . . . 1
1.3 Extensibility . . . 1

2 Application of ML Pipeline to Financial Distress Prediction **2**
2.1 Background . . . 2
2.2 Summary Statistics . . . 2
2.3 Data Preparation . . . 4
2.4 Feature Choices . . . 5
2.5 Model Choices . . . 5
2.6 Results . . . 5

Notes

- Representative code snippets are interspersed with analysis and explanations below; all code is available on GitHub: `https://github.com/satejsoman/capp30254/tree/master/hw2/code`.
- The pipeline library lives in the `code/pipeline` directory, while the sample application which imports the library is `code/distress-classifier.py`.

1 Pipeline Library Design

1.1 Overview

The `pipeline` library is a Python utility for declaratively creating sequences of data transformations and machine learning test/training routines. The two main classes are:

- `Pipeline`, which specifies input data sources, output directories, transformation sequences, and machine learning model parameters.
- `Transformation`, a wrapper over a Python callable, annotated with a human-readable name, and a declaration of input and output columns. In implementation, there is no difference between generating new columns in data cleaning/processing and generating feature vectors, so both stages are represented by collections of `Transformation` objects.

1.2 Design Decisions

The `pipeline` library aims for:

- **reproducibility**: each pipeline run logs transformation steps, model parameters, pipeline library version, and input data hashes in order to create uniquely identifiable output artifacts.
- **debuggability**: each step is logged to `stdout` and a persistent file in a human readable manner so end users can understand the state of the pipeline.
- **customization**: the declarative syntax for `Pipeline` objects is flexible enough to chain machine learning pipelines or include multiple stages of preprocessing and feature generation.

To a large extent, `Pipeline` is a wrapper over on Pandas `DataFrame` objects.

1.3 Extensibility

To customize functionality, the `Pipeline` class can be subclassed, or individual `Pipeline` instance can have bound methods replaced by application-level code (e.g. "monkey-patching"). The example application in section 2 uses the latter approach.

Additionally, by rigorously tracking transformation sequences, input data, and model parameters, the library is, in principle, able to support caching of steps (i.e. no need to recompute features if they have not changed), though this feature has not been implemented at time of writing.

2 Application of ML Pipeline to Financial Distress Prediction

2.1 Background

When making a decision on whether to lend to an individual, banks and financial institutions have access to a number of demographic and financial variables for each prospective borrower. The lenders would like to know, at decision time, the risk that a loan will not be repaid. With historical financial delinquency information and the `pipeline` library, we can use these demographic and financial data to build a model to predict whether a person is a serious risk for non-repayment of a loan.

2.2 Summary Statistics

In order to generate the summary statistics, we can write a custom summary function and attach it to our pipeline:

```
from types import MethodType

def summarize_fd_data(self):
    self.logger.info("Running custom summary function")
    df = self.dataframe
    summary = (df
        .describe(percentiles=[])
        .drop("count")
        .drop("50%")
        .append(pd.DataFrame(
            [df.corr()[self.target].apply(np.abs), df.isnull().sum()],
            index=["abs corr", "missing"]))
        .T
        .rename(dict(zip(summary.index, clean_names))
        .sort_values("abs corr", ascending=False))

    with (self.output_dir/"summary.tex").open('w') as fer:
        summary.to_latex(buf=fer, float_format="%.2f")

    for column in df.columns, colors:
        df[[column]].plot.hist(bins=20)
        matplotlib2tikz.save(self.output_dir/(column + ".tex"), figureheight="3in",
            figurewidth="3in")
    return self


pipeline = Pipeline(...)
pipeline.summarize_data = MethodType(summarize_fd_data, pipeline)
```

	mean	std	min	max	abs corr	missing
delinquency	0.16	0.37	0.00	1.00	1.00	0.00
id	115800.15	28112.72	22.00	149999.00	0.62	0.00
age	51.68	14.75	21.00	109.00	0.17	0.00
payments 30-59 days late	0.59	5.21	0.00	98.00	0.15	0.00
payments 90 days late	0.42	5.19	0.00	98.00	0.14	0.00
payments 60-89 days late	0.37	5.17	0.00	98.00	0.12	0.00
number of dependents	0.77	1.12	0.00	13.00	0.07	1037.00
zipcode	60623.82	11.98	60601.00	60644.00	0.05	0.00
number of credit lines	8.40	5.21	0.00	56.00	0.04	0.00
monthly income	6579.00	13446.83	0.00	1794060.00	0.03	7974.00
debt ratio	331.46	1296.11	0.00	106885.00	0.01	0.00
number of real estate loans	1.01	1.15	0.00	32.00	0.01	0.00
revolving utilization	6.38	221.62	0.00	22000.00	0.00	0.00

Table 1: Table of relevant summary statistics.

The following figures show the distributions of some sample variables:

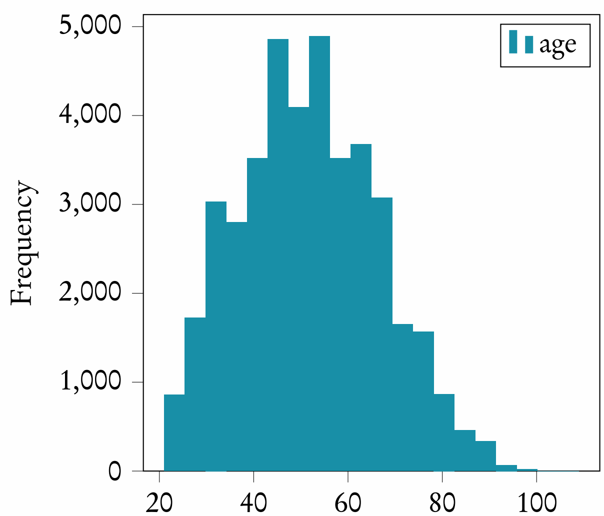

Figure 1: Distribution of age

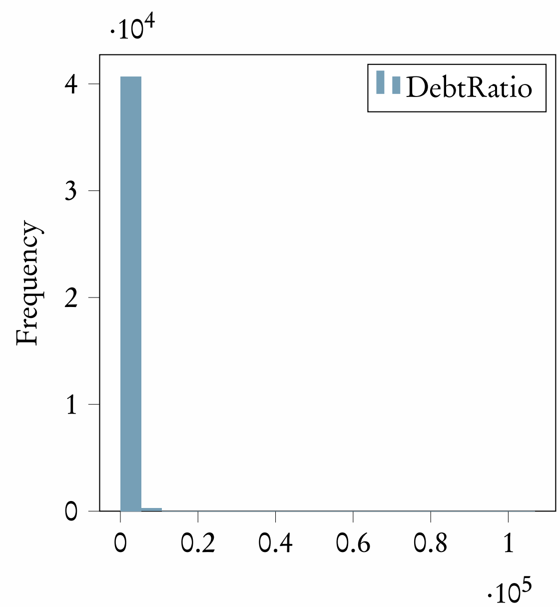

Figure 2: Distribution of DebtRatio

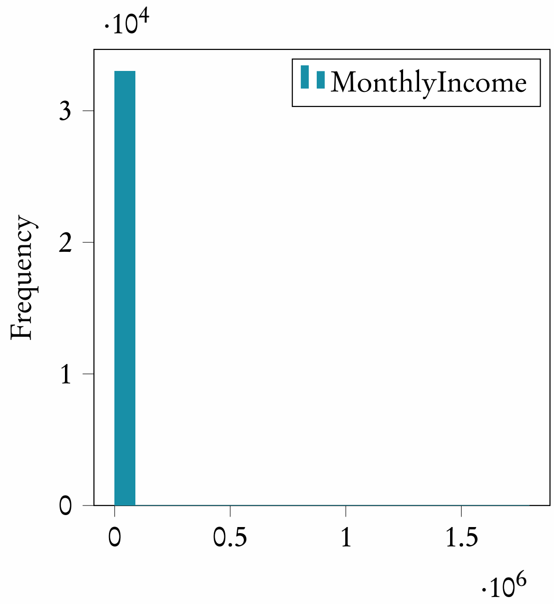

Figure 3: Distribution of `MonthlyIncome`

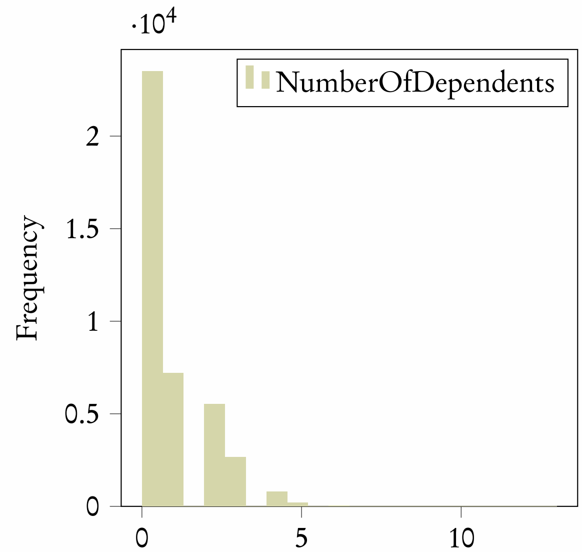

Figure 4: Distribution of `NumberOfDependents`

2.3 Data Preparation

As seen in Table 1, the columns for number of dependents and monthly income have missing values. We can clean the data by replacing the missing values in these columns by the average of the non-missing values.

```
from pipeline.transformation import replace_missing

pipeline = Pipeline(...
    data_preprocessors=[
        replace_missing("MonthlyIncome"),
        replace_missing("NumberOfDependents")],
    ...)
```

2.4 Feature Choices

The variables in Table 1 are sorted by their absolute correlation with delinquency ($|\rho_{x,y}|$). Since the person's ID is likely randomly assigned, its high correlation with delinquency is spurious. Looking at the next few variables, the combination of age, along with *any* payments late past 30 days, show promise of explaining delinquency well. We can also include income in our model; standard econometrics practice is to analyze the effects of changes in logarithmic income since the income distribution tends to be skewed.

```
from pipeline.transformation import Transformation, replace_missing

# categorical variable
age_decade = Transformation("age-decade", ["age"], "age_decade",
    lambda col: col.apply(lambda x: 10*(x//10))
)

# binary variable
any_late_payments = Transformation("any-late-payments",
    ["NumberOfTime30-59DaysPastDueNotWorse", "NumberOfTime60-89DaysPastDueNotWorse",
    "NumberOfTimes90DaysLate"], "any_late_payments",
    lambda cols: cols.apply(lambda x: int(np.sum(x) > 0), axis=1)
)

log_monthly_income = Transformation("log-monthly-income", ["MonthlyIncome_clean"],
    "log_monthly_income",
    lambda col: col.apply(lambda x: np.log(x + 1e-10))
)

pipeline = Pipeline(...
    feature_generators=[
        age_decade,
        any_late_payments,
        log_monthly_income],
    ...)
```

2.5 Model Choices

A logistic regression is appropriate here:

$$P(\text{delinquency} \mid \text{data}) = \Lambda\big(\beta_0 + \beta_1(\text{any late payments}) + \beta_2(\text{age decade}) + \beta_3 \cdot \text{LOG}(\text{monthly income})\big)$$

$\Lambda(w)$ is the logistic function. For this application, we use the implementation found in the scikit-learn library (`sklearn.linear_model.LogisticRegression`).

2.6 Results

We choose accuracy as the evaluation metric since this exercise is about the pipeline rather than the technique. Running the pipeline shows:

```
# set up pipeline
pipeline = Pipeline(input_path, "SeriousDlqin2yrs",
    summarize=True,
    data_preprocessors=[
        replace_missing("MonthlyIncome"),
        replace_missing("NumberOfDependents")],
    feature_generators=[
        age_decade,
        any_late_payments,
        log_monthly_income],
    model=LogisticRegression(solver="lbfgs"),
    name="financial-distress-classifier",
    output_root_dir="output")

# attach custom summary function
pipeline.summarize_data = MethodType(summarize_fd_data, pipeline)

# run pipeline
pipeline.run()
```

The relevant portion of the pipeline output is shown:

```
Running transformations for preprocessing
    Applying transformation (1/2): replace-missing-values-with-mean(MonthlyIncome)
    ['MonthlyIncome'] -> MonthlyIncome_clean
    Applying transformation (2/2): replace-missing-values-with-mean(NumberOfDependents)
    ['NumberOfDependents'] -> NumberOfDependents_clean


Running transformations for feature generation
    Applying transformation (1/3): age-decade
    ['age'] -> age_decade
    Applying transformation (2/3): any-late-payments
    ['NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTime60-89DaysPastDueNotWorse',
        'NumberOfTimes90DaysLate'] -> any_late_payments
    Applying transformation (3/3): log-monthly-income
    ['MonthlyIncome_clean'] -> log_monthly_income

Running model LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
Features: ['age_decade', 'any_late_payments', 'log_monthly_income']
Fitting: SeriousDlqin2yrs
Evaluating model
Model score: 0.8415008777062609
Copying artifacts to stable path
Finished at 2019-04-18 00:18:47.284784
```

As we can see in the pipeline output, the model score accuracy is 0.8415. This is not a fantastic score, but we have a flexible framework to quickly and effectively increase the performance of the classifier.