Why generate features?

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Feature Engineering

House A is a **two** bedroomed house **2000** sq. ft brownstone.

House B is **1500** sq. ft with **one** bedroom.

House	Bedrooms	sq. ft
A	2	2000
B	1	1500
...	...	...

Feature Engineering is the act of taking raw data and extracting features from it that are suitable for tasks like machine learning.

Most machine learning algorithms work with tabular data

Features = information stored in the columns of this tabular data.

In this example, the features would be bedrooms and sq. ft. of each house.

Most machine learning algorithms require their input data to be represented as a vector or a matrix, and many assume that the data is distributed normally.

Feature engineering is often overlooked in machine learning discussions, but any real-world practitioner will confirm that data manipulation and feature engineering is the most important aspect of the proect.

Different types of data

- Continuous: either integers (or whole numbers) or floats (decimals)
- Categorical: one of a limited set of values, e.g. gender, country of birth
- Ordinal: ranked values, often with no detail of distance between them
- Boolean: True/False values
- Datetime: dates and times

Course structure

- Chapter 1: Feature creation and extraction
- Chapter 2: Engineering messy data
- Chapter 3: Feature normalization
- Chapter 4: Working with text features

The first chapter has you ingest and create basic features from tabular data.

In the second chapter, you deal with data that has missing values.

The third chapter has you transforming your data so that it conforms to statistical assumptions often necessary for machine learning models.

Finally, in the last chapter, you'll convert free form text into tabular data so it can be used with machine learning models.

Pandas

```
import pandas as pd
df = pd.read_csv(path_to_csv_file)
print(df.head())
```

Dataset

```
             SurveyDate  \
0   2018-02-28 20:20:00
1   2018-06-28 13:26:00
2   2018-06-06 03:37:00
3   2018-05-09 01:06:00
4   2018-04-12 22:41:00


                          FormalEducation
0   Bachelor's degree (BA. BS. B.Eng.. etc.)
1   Bachelor's degree (BA. BS. B.Eng.. etc.)
2   Bachelor's degree (BA. BS. B.Eng.. etc.)
3   Some college/university study  ...
4   Bachelor's degree (BA. BS. B.Eng.. etc.)
```

Column names

```
print(df.columns)
```

```
Index(['SurveyDate', 'FormalEducation',
       'ConvertedSalary', 'Hobby', 'Country',
       'StackOverflowJobsRecommend', 'VersionControl',
       'Age', 'Years Experience', 'Gender',
       'RawSalary'], dtype='object')
```

Column types

```
print(df.dtypes)
```

```
SurveyDate                    object
FormalEducation               object
ConvertedSalary              float64
...
Years Experience               int64
Gender                        object
RawSalary                     object
dtype: object
```

Selecting specific data types

```
only_ints = df.select_dtypes(include=['int'])
print(only_ints.columns)
```

```
Index(['Age', 'Years Experience'], dtype='object')
```

```
# Import pandas
import pandas as pd

# Import so_survey_csv into so_survey_df
so_survey_df = pd.read_csv(so_survey_csv)

# Print the first five rows of the DataFrame
print(so_survey_df.head())

# Print the data type of each column
print(so_survey_df.dtypes)
```

Question

What type of data is the `ConvertedSalary` column?

Possible Answers

- Datetime
- Numeric
- String
- Boolean

Submit Answer

Take Hint (-15 XP)

IPython Shell | Slides

```
    7    Male  £41,671.00

[5 rows x 11 columns]
SurveyDate                     object
FormalEducation                object
ConvertedSalary               float64
Hobby                          object
Country                        object
StackOverflowJobsRecommend    float64
VersionControl                 object
Age                             int64
Years Experience                int64
Gender                         object
RawSalary                      object
```

Lets get going!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

```
# Create subset of only the numeric columns
so_numeric_df = so_survey_df.select_dtypes(include=['int','float'])

# Print the column names contained in so_survey_df_num
print(so_numeric_df.columns)
```

Dealing with Categorical Variables

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Encoding categorical features

Index	Country
1	‘India’
2	’USA’
3	‘UK’
4	‘UK’
5	‘France’
...	...

Categorical variables are used to represent groups that are qualitative in nature.

Examples are colors, like blue, red, etc.

While these can be easily understood by humans, for machine learning model purposes, thsy need to be encoded as numerical values.

If you naively assign the order of India, 1, USA 2, etc. that's categorizing them in an unordered manner, and it may greatly penalize the effectiveness ofthe machine learninig model.

Encoding categorical features

Instead, the values can be encoded by creating additional binary features corresponding to whether each value was picked or not as shown in the table on the left.

This leverages the information of what country is given, without inferring any order between the different options.

Index	Country
1	'India'
2	'USA'
3	'UK'
4	'UK'
5	'France'
...	...

Index	C_India	C_USA	C_UK	C_France
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	1	0
5	0	0	0	1
...	...	...	...	...

These are very similar and often confused. By default, pandas performs one-hot encoding when you use get_dummies() function.

Encoding categorical features

- One-hot encoding
- Dummy encoding

One-hot encoding

One-hot encoding converts n categories into n features as shown below. The function get_dummies() takes a dataframe and a list of categorical columns you want converted into one-hot encoded columns, and returns an updated dataframe with these columns included.

Specifying a prefix with the prefix argument can improve readability like the letter C for country which has been used below.

```
pd.get_dummies(df, columns=['Country'],
               prefix='C')
```

```
   C_France  C_India  C_UK  C_USA
0         0        1     0      0
1         0        0     0      1
2         0        0     1      0
3         0        0     1      0
4         1        0     0      0
```

Dummy encoding

Dummy encoding creates n-1 ffeatures for n categories, omitting the first category. For example, here there's no column for France.

In dummy encoding, the base value, France in this case, is encoded by the absence of all other counries, as you see on the last row where the value is represented by the intercept.

Dummy encoding is distinguished form one-hot encoding by the addition of the drop_first argument in pandas.

```
pd.get_dummies(df, columns=['Country'],
               drop_first=True, prefix='C')
```

```
   C_India  C_UK  C_USA
0        1     0      0
1        0     0      1
2        0     1      0
3        0     1      0
4        0     0      0
```

One-hot vs. dummies

One-hot encoding generally creates much more explainable features, as each country will have its own weight that can be observed after training.

One-hot encoding may create features that are entirely collinear due to the same information being represented multiple times.

- **One-hot encoding**: Explainable features
- **Dummy encoding**: Necessary information without duplication

Index	Sex
0	Male
1	Female
2	Male

Having the double representation of both Male & Female, when just recodring a 0 in one column, say Male, marks them as Female.

The double representation can lead to instability in your models and dummy values would be more appropriate

Index	Male	Female
0	1	0
1	0	1
2	1	0

Index	Male
0	1
1	0
2	1

Limiting your columns

Both one-hot encoding and dummy encoding may result in a huge number of columns being created if there are too many different categories in a column.

In those cases, only create columns for the most common values. YOu can check the number of values by using the value_counts() function on a specific column.

```
counts = df['Country'].value_counts()
print(counts)
```

```
'USA'       8
'UK'        6
'India'     2
'France'    1
Name: Country, dtype: object
```

Limiting your columns

Once you have your counts, you can use it to limit what values you will include by first creating a mask of the values that occur less than n times.

A mask is a list of booleans outlining which values in a column should be affected.

First we find the categories that occue less than n times using the index attribute and wrap this inside the isin() method.

After you create the mask, you can use it to replace these categories that occur less than n time with a value of you choice as shown here.

```
mask = df['Country'].isin(counts[counts < 5].index)

df['Country'][mask] = 'Other'

print(pd.value_counts(colors))
```

```
'USA'      8
'UK'       6
'Other'    3
Name: Country, dtype: object
```

```
# Convert the Country column to a one hot encoded Data Frame
one_hot_encoded = pd.get_dummies(so_survey_df, columns = ['Country'], prefix='OH')

# Print the columns names
print(one_hot_encoded.columns)

# Create dummy variables for the Country column
dummy = pd.get_dummies(so_survey_df, columns=['Country'], drop_first = True, prefix='DM')

# Print the columns names
print(dummy.columns)
```

Did you notice that the column for France was missing when you created dummy variables?
Now you can choose to use one-hot encoding or dummy variables where appropriate.

```
# Create a series out of the Country column
countries = so_survey_df.Country

# Get the counts of each category
country_counts = countries.value_counts()

# Create a mask for only categories that occur less than 10 times
mask = countries.isin(country_counts[country_counts < 10].index)

# Label all other categories as Other
countries[mask] = 'Other'

# Print the updated category counts
print(countries.value_counts())
```

Now you deal with categorical variables

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Numeric variables

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

Robert O'Callaghan
Director of Data Science, Ordergroove

Types of numeric features

- Age
- Price
- Counts
- Geospatial data

Does size matter?

One of the first questions you should ask when working with numeric features is whether the magnitude of the feature is its most important trait, or just it's direction.

In this example, you might care more about whether a restaurant had any major violations at all, over whether it was a repeat offender.

	Resturant_ID	Number_of_Violations
0	RS_1	0
1	RS_2	0
2	RS_3	2
3	RS_4	1
4	RS_5	0
5	RS_6	0
6	RS_7	4
7	RS_8	4
8	RS_9	1
9	RS_10	0

Binarizing numeric variables

```
df['Binary_Violation'] = 0

df.loc[df['Number_of_Violations'] > 0,
       'Binary_Violation'] = 1
```

Binarizing numeric variables

	Resturant_ID	Number_of_Violations	Binary_Violation
0	RS_1	0	0
1	RS_2	0	0
2	RS_3	2	1
3	RS_4	1	1
4	RS_5	0	0
5	RS_6	0	0
6	RS_7	4	1
7	RS_8	4	1
8	RS_9	1	1
9	RS_10	0	0

Binning numeric variables

```
import numpy as np
df['Binned_Group'] = pd.cut(
    df['Number_of_Violations'],
    bins=[-np.inf, 0, 2, np.inf],
    labels=[1, 2, 3]
)
```

Binning numeric variables

	Resturant_ID	Number_of_Violations	Binned_Group
0	RS_1	0	1
1	RS_2	0	1
2	RS_3	2	2
3	RS_4	1	2
4	RS_5	0	1
5	RS_6	0	1
6	RS_7	4	3
7	RS_8	4	3
8	RS_9	1	2
9	RS_10	0	1

While numeric values can often be used without any feature engineering, there will be cases when some form of manipulation can be useful. For example on some occasions, you might not care about the magnitude of a value but only care about its direction, or if it exists at all. In these situations, you will want to binarize a column. In the so_survey_df data, you have a large number of survey respondents that are working voluntarily (without pay). You will create a new column titled Paid_Job indicating whether each person is paid (their salary is greater than zero).

```
# Create the Paid_Job column filled with zeros
so_survey_df['Paid_Job'] = 0

# Replace all the Paid_Job values where ConvertedSalary is > 0
so_survey_df.loc[so_survey_df.ConvertedSalary > 0, 'Paid_Job'] = 1

# Print the first five rows of the columns
print(so_survey_df[['Paid_Job', 'ConvertedSalary']].head())
```

```
# Import numpy
import numpy as np

# Specify the boundaries of the bins
bins = [-np.inf, 10000, 50000, 100000, 150000, np.inf]

# Bin labels
labels = ['Very low', 'Low', 'Medium', 'High', 'Very high']

# Bin the continuous variable ConvertedSalary using these boundaries
so_survey_df['boundary_binned'] = pd.cut(so_survey_df['ConvertedSalary'],
                                         bins = bins, labels = labels)

# Print the first 5 rows of the boundary_binned column
print(so_survey_df[['boundary_binned', 'ConvertedSalary']].head())
```

Lets start practicing!

FEATURE ENGINEERING FOR MACHINE LEARNING IN PYTHON

For many continuous values you will care less about the exact value of a numeric column, but instead care about the bucket it falls into. This can be useful when plotting values, or simplifying your machine learning models. It is mostly used on continuous variables where accuracy is not the biggest concern e.g. age, height, wages.

Bins are created using pd.cut(df['column_name'], bins) where bins can be an integer specifying the number of evenly spaced bins, or a list of bin boundaries.

```
# Bin the continuous variable ConvertedSalary into 5 bins
so_survey_df['equal_binned'] = pd.cut(so_survey_df['ConvertedSalary'], bins = 5)

# Print the first 5 rows of the equal_binned column
print(so_survey_df[['equal_binned', 'ConvertedSalary']].head())
```