CUSTOMER ANALYTICS: PREPARING DATA FOR MODELLING

Omotola Ayodele Lawal

2024-09-16

Table of contents

Project Overview

Figure 1: Two data scientists working on a dashboard.

A common problem when creating models to generate business value from data is that the datasets can be so large that it can take days for the model to generate predictions. Ensuring that your dataset is stored as efficiently as possible is crucial for allowing these models to run on a more reasonable timescale without having to reduce the size of the dataset.

You've been hired by a major online data science training provider called *Training Data Ltd.* to clean up one of their largest customer datasets. This dataset will eventually be used to predict whether their students are looking for a new job or not, information that they will then use to direct them to prospective recruiters.

You've been given access to `customer_train.csv`, which is a subset of their entire customer dataset, so you can create a proof-of-concept of a much more efficient storage solution. The dataset contains anonymized student information, and whether they were looking for a new job or not during training:

Column	Description
`student_id`	A unique ID for each student.
`city`	A code for the city the student lives in.
`city_development_index`	A scaled development index for the city.
`gender`	The student's gender.
`relevant_experience`	An indicator of the student's work relevant experience.
`enrolled_university`	The type of university course enrolled in (if any).
`education_level`	The student's education level.
`major_discipline`	The educational discipline of the student.
`experience`	The student's total work experience (in years).
`company_size`	The number of employees at the student's current employer.
`company_type`	The type of company employing the student.
`last_new_job`	The number of years between the student's current and previous jobs.
`training_hours`	The number of hours of training completed.
`job_change`	An indicator of whether the student is looking for a new job (`1`) or not (`0`).

Task

The Head Data Scientist at Training Data Ltd. has asked you to create a `DataFrame` called `ds_jobs_transformed` that stores the data in `customer_train.csv` much more efficiently. Specifically, they have set the following requirements:

- Columns containing categories with only two factors must be stored as Booleans (bool).
- Columns containing integers only must be stored as 32-bit integers (int32).
- Columns containing floats must be stored as 16-bit floats (float16).
- Columns containing nominal categorical data must be stored as the category data type.
- Columns containing ordinal categorical data must be stored as ordered categories, and not mapped to numerical values, with an order that reflects the natural order of the column.

- The DataFrame should be filtered to only contain students with 10 or more years of experience at companies with at least 1000 employees, as their recruiter base is suited to more experienced professionals at enterprise companies.

> **Important**
>
> If you call `.info()` or `.memory_usage()` methods on `ds_jobs` and `ds_jobs_transformed` after you've preprocessed it, you should notice a substantial decrease in memory usage.

Data Source

Data: The primary data used for this analysis is the `customer_train.csv`, which is a subset of the entire customer dataset

Tools

Jupyter lab

Steps/Explanations

- The necessary library was imported, which is `Pandas`
- The Original dataset was loaded, named `ds_jobs` and a copy was made, called `ds_jobs_transformed`
- Exploratory Data Analysis was performed which help identify ordinal, nominal, and two-factor categories. This was done by written codes, which iterate over all columns of the DataFrame `ds_jobs` that have a data type of object (typically representing strings or categorical data) and print the value counts of each column.
- A dictionary of columns containing ordered categorical data was created. The code defines the dictionary called ordered_cats, which contains lists of ordered categories for various features. These features represent specific categorical data in a dataset (e.g., levels of education, size of the company, work experience, etc.). This dictionary can later be used to create ordered categorical columns, for example, when transforming or encoding data in a pandas DataFrame.
- A mapping dictionary of columns containing two-factor categories to convert to Booleans was created. The code defines a Python dictionary called two_factor_cats. This dictionary is used to map certain categorical values into Boolean (True or False) values.
- This `for col in ds_jobs_transformed:` code iterates through each column in the `ds_jobs_transformed` DataFrame, performing different transformations based on the column name.

- For the columns `'relevant_experience'` and `'job_change'`, if col in `['relevant_experience', 'job_change']:` code uses the `two_factor_cats` dictionary to convert their categorical values into boolean values (True/False).
- The .map() function applies the mapping from two_factor_cats (defined previously) to the column. For example:
  - `'No relevant experience'` becomes `False`.
  - `'Has relevant experience'` becomes `True`.
  - `0.0` becomes `False`.
  - `1.0` becomes `True`.
- For the columns `'student_id'` and `'training_hours'`, `elif col in ['student_id', 'training_hours']:` code changes their data types to int32 using `.astype('int32')`. This helps reduce memory usage, especially if the original data type was int64. The smaller int32 uses less memory and is sufficient for storing integers that fit within the 32-bit range.
- The column `'city_development_index'` is converted to `float16 (16-bit floating-point format)` by the following code:

```
elif col in ['student_id', 'training_hours']:
        ds_jobs_transformed[col] = ds_jobs_transformed[col].astype('int32')
```

- `float16` consumes less memory than the default `float64`.
- It's useful for reducing the memory footprint of large datasets when the precision provided by `float16` is sufficient.
- For columns that are in the `ordered_cats` dictionary created earlier, the code below converts them to ordered categorical data types.

```
elif col in ordered_cats.keys():
    category = pd.CategoricalDtype(ordered_cats[col], ordered=True)
    ds_jobs_transformed[col] = ds_jobs_transformed[col].astype(category)
```

- `ordered_cats` is a dictionary that contains the order of categories for certain columns.
- `pd.CategoricalDtype` creates a categorical data type with a specific order, which is useful when the categories have a meaningful order (e.g., educational levels or experience).
- `ordered=True` ensures that the categories are treated as ordered (e.g., "Primary School" < "High School" < "Graduate").
- The `.astype(category)` applies this conversion.
- For all remaining columns (those not handled in the previous conditions), the code below converts them to the standard categorical data type without an explicit order.

```
else:
    ds_jobs_transformed[col] = ds_jobs_transformed[col].astype('category')
```

- Converting to category reduces memory usage, especially when the column has a limited number of distinct values (e.g., city names, job roles, etc.).
- The final DataFrame was filtered to only contain students with 10 or more years of experience at companies with at least 1000 employees, as their recruiter base is suited to more experienced professionals at enterprise companies.

```python
# Import necessary libraries
import pandas as pd

# Load the dataset
ds_jobs = pd.read_csv("customer_train.csv")

# View the dataset
ds_jobs.head()

ds_jobs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19158 entries, 0 to 19157
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   student_id              19158 non-null  int64  
 1   city                    19158 non-null  object 
 2   city_development_index  19158 non-null  float64
 3   gender                  14650 non-null  object 
 4   relevant_experience     19158 non-null  object 
 5   enrolled_university     18772 non-null  object 
 6   education_level         18698 non-null  object 
 7   major_discipline        16345 non-null  object 
 8   experience              19093 non-null  object 
 9   company_size            13220 non-null  object 
 10  company_type            13018 non-null  object 
 11  last_new_job            18735 non-null  object 
 12  training_hours          19158 non-null  int64  
 13  job_change              19158 non-null  float64
dtypes: float64(2), int64(2), object(10)
memory usage: 2.0+ MB
```

Data Analysis

Include below are the codes used to achieve the task given

```python
# Import necessary libraries
import pandas as pd

# Load the dataset and create a copy
ds_jobs = pd.read_csv("customer_train.csv")
ds_jobs_transformed = ds_jobs.copy()

# EDA to help identify ordinal, nominal, and two-factor categories
for col in ds_jobs.select_dtypes("object").columns:
    print(ds_jobs_transformed[col].value_counts(), '\n')

# Create a dictionary of columns containing ordered categorical data
ordered_cats = {
    'enrolled_university': ['no_enrollment', 'Part time course', 'Full time course'],
    'education_level': ['Primary School', 'High School', 'Graduate', 'Masters', 'Phd'],
    'experience': ['<1'] + list(map(str, range(1, 21))) + ['>20'],
    'company_size': ['<10', '10-49', '50-99', '100-499', '500-999', '1000-4999', '5000-9999', '10
    'last_new_job': ['never', '1', '2', '3', '4', '>4']
}

# Create a mapping dictionary of columns containing two-factor categories to convert to Booleans
two_factor_cats = {
    'relevant_experience': {'No relevant experience': False, 'Has relevant experience': True},
    'job_change': {0.0: False, 1.0: True}
}

# Loop through DataFrame columns to efficiently change data types
for col in ds_jobs_transformed:

    # Convert two-factor categories to bool
    if col in ['relevant_experience', 'job_change']:
        ds_jobs_transformed[col] = ds_jobs_transformed[col].map(two_factor_cats[col])

    # Convert integer columns to int32
    elif col in ['student_id', 'training_hours']:
        ds_jobs_transformed[col] = ds_jobs_transformed[col].astype('int32')

    # Convert float columns to float16
    elif col == 'city_development_index':
        ds_jobs_transformed[col] = ds_jobs_transformed[col].astype('float16')

    # Convert columns containing ordered categorical data to ordered categories using dict
    elif col in ordered_cats.keys():
        category = pd.CategoricalDtype(ordered_cats[col], ordered=True)
        ds_jobs_transformed[col] = ds_jobs_transformed[col].astype(category)
```

```python
    # Convert remaining columns to standard categories
    else:
        ds_jobs_transformed[col] = ds_jobs_transformed[col].astype('category')

# Filter students with 10 or more years experience at companies with at least 1000 employees
ds_jobs_transformed = ds_jobs_transformed[(ds_jobs_transformed['experience'] >= '10') & (ds_jobs_

ds_jobs_transformed.info()
```

```
city
city_103    4355
city_21     2702
city_16     1533
city_114    1336
city_160     845
            ...
city_129       3
city_111       3
city_121       3
city_140       1
city_171       1
Name: count, Length: 123, dtype: int64

gender
Male      13221
Female     1238
Other       191
Name: count, dtype: int64

relevant_experience
Has relevant experience    13792
No relevant experience      5366
Name: count, dtype: int64

enrolled_university
no_enrollment       13817
Full time course     3757
Part time course     1198
Name: count, dtype: int64

education_level
Graduate          11598
Masters            4361
High School        2017
```

```
Phd                 414
Primary School      308
Name: count, dtype: int64

major_discipline
STEM               14492
Humanities           669
Other                381
Business Degree      327
Arts                 253
No Major             223
Name: count, dtype: int64

experience
>20    3286
5      1430
4      1403
3      1354
6      1216
2      1127
7      1028
10      985
9       980
8       802
15      686
11      664
14      586
1       549
<1      522
16      508
12      494
13      399
17      342
19      304
18      280
20      148
Name: count, dtype: int64

company_size
50-99        3083
100-499      2571
10000+       2019
10-49        1471
1000-4999    1328
<10          1308
```

```
500-999       877
5000-9999     563
Name: count, dtype: int64

company_type
Pvt Ltd               9817
Funded Startup        1001
Public Sector          955
Early Stage Startup    603
NGO                    521
Other                  121
Name: count, dtype: int64

last_new_job
1        8040
>4       3290
2        2900
never    2452
4        1029
3        1024
Name: count, dtype: int64

<class 'pandas.core.frame.DataFrame'>
Index: 2201 entries, 9 to 19143
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   student_id              2201 non-null   int32
 1   city                    2201 non-null   category
 2   city_development_index  2201 non-null   float16
 3   gender                  1821 non-null   category
 4   relevant_experience     2201 non-null   bool
 5   enrolled_university     2185 non-null   category
 6   education_level         2184 non-null   category
 7   major_discipline        2097 non-null   category
 8   experience              2201 non-null   category
 9   company_size            2201 non-null   category
 10  company_type            2144 non-null   category
 11  last_new_job            2184 non-null   category
 12  training_hours          2201 non-null   int32
 13  job_change              2201 non-null   bool
dtypes: bool(2), category(9), float16(1), int32(2)
memory usage: 69.5 KB
```

Results

There was a marked reduction in the size of the `ds_jobs_transformed` DataFrame from 2.0+ MB to 69.5 KB, which help to save memory and prepare the data for predictive modelling.

Recommendations

None

Limitations

None

References

1. Filtering DataFrames in Intermediate Python Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by Hugo Bowne-Henderson.
2. For loop in Intermediate Python Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by Hugo Bowne-Henderson.
3. Python For Data Analysis 3E (Online) by Wes Mckinney Click here to preview
4. Working with Categorical Data in Python in Intermediate Python Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by Kasey Jones.