ML_PipeLines (3)

March 27, 2025

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

[2]:
```
from google.colab import auth
auth.authenticate_user()

import gspread
from google.auth import default
creds, _ = default()

gc = gspread.authorize(creds)

worksheet = gc.open('cereal-kaggle').sheet1

# get_all_values gives a list of rows.
rows = worksheet.get_all_values()
# print(rows)

# Convert to a DataFrame and render.
import pandas as pd
df = pd.DataFrame.from_records(rows)
```

0.0.1 Cleaning the data

[3]:
```
# setting first row as headers
# resetting index
# didplaying first 5 rows
df.columns = df.iloc[0]
df = df.iloc[1:].reset_index(drop=True)
df.head()
```

[3]:
```
0                       name mfr type calories protein fat sodium fiber carbo  \
0                  100% Bran   N    C                4   1    130    10     5
1          100% Natural Bran   Q    C      120       3   5     15     2     8
2                   All-Bran   K    C       70       4   1    260     9     7
3  All-Bran with Extra Fiber   K    C       50       4   0    140    14     8
4             Almond Delight   R    C                2   2    200     1    14

0 sugars potass vitamins shelf weight  cups     rating
0      6    280       25   top      1  0.33  68.402973
1      8    135        0   top      1     1  33.983679
2      5    320       25   top      1  0.33  59.425505
3      0    330       25   top      1   0.5  93.704912
4      8     -1       25            1  0.75  34.384843
```

[4]:
```
# mfr, type, calories, protein, fat, fiber, sugars, shelf
df_list = ['mfr', 'type', 'calories', 'protein', 'fat', 'fiber', 'sugars',␣
 ↪'shelf', 'rating']
df = df[df_list]
df.head()
```

[4]:
```
0 mfr type calories protein fat fiber sugars shelf     rating
0   N    C                4   1    10      6   top  68.402973
1   Q    C      120       3   5     2      8   top  33.983679
2   K    C       70       4   1     9      5   top  59.425505
3   K    C       50       4   0    14      0   top  93.704912
4   R    C                2   2     1      8        34.384843
```

[5]:
```
# Checking for missing values
df.isnull().sum().sum()
```

[5]:
```
np.int64(0)
```

[6]:
```
# checking for empty strings
for col in df.columns:
  print(col, (df[col] == '').sum())
```

```
mfr 2
type 0
calories 5
```

```
protein 0
fat 7
fiber 6
sugars 6
shelf 2
rating 0
```

[7]:
```
# Replacing empty strings with np.nan
df.replace('', np.nan, inplace=True)
```

```
<ipython-input-7-5ffaf50ca23e>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.replace('', np.nan, inplace=True)
```

0.0.2 Define target and features and train-test-split:

[8]:
```
# Define features (X) and target (y).
x = df.drop(['rating'], axis=1)
y = df['rating']
```

[9]:
```
# Train test split the data to prepare for machine learning (use a random state␣
↪of 42 for reproducibility).
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)
```

0.0.3 Create 3 pipelines (one for ordinal, categorical, and numeric features).

For the ordinal pipeline:

[10]:
```
# Save a list of ordinal features
ordinal_features = ['shelf']
```

[11]:
```
# Impute null values using SimpleImputer using the "most_frequent" strategy.
# Instantisting SimpleImputer
impute = SimpleImputer(strategy='most_frequent')
x[ordinal_features] = impute.fit_transform(x[ordinal_features])
```

[12]:
```
x[ordinal_features].head()
```

[12]:
```
0 shelf
0   top
1   top
2   top
3   top
4   top
```

[13]:
```
# Use OrdinalEncoder to encode the "shelf" column.
# Instantiating
ordinal_encoder = OrdinalEncoder()
x[ordinal_features] = ordinal_encoder.fit_transform(x[ordinal_features])
```

[14]:
```
# Scale the ordinal features using StandardScaler
# Instantiating
scaler = StandardScaler()
x[ordinal_features] = scaler.fit_transform(x[ordinal_features])
```

[15]:
```
# Display the pipeline to confirm the code was error-free
x[ordinal_features].head()
```

[15]:
```
0     shelf
0  0.937043
1  0.937043
2  0.937043
3  0.937043
4  0.937043
```

For categorical (nominal) pipeline:

[16]:
```
cat_features = ['mfr', 'type']
x[cat_features].head()
```

[16]:
```
0 mfr type
0   N    C
1   Q    C
2   K    C
3   K    C
4   R    C
```

[17]:
```
# Impute null values using SimpleImputer using the ‘constant’ strategy with a␣
  ↪fill value of "MISSING."
impute = SimpleImputer(strategy='constant', fill_value='MISSING')
x[cat_features] = impute.fit_transform(x[cat_features])
```

[18]:
```
# Use OneHotEncoder to encode the features
one_hot_encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_cat_features = one_hot_encoder.fit_transform(x[cat_features])
```

[19]:
```
# Display the pipeline to confirm the code was error-free
encoded_cat_features[:5]
```

[19]:
```
array([[0., 0., 0., 0., 1., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 1., 0.]])
```

For the numeric features pipeline:

[20]:
```
df.head(2)
```

[20]:
```
0 mfr type calories protein fat fiber sugars shelf     rating
0   N    C      NaN       4   1    10      6   top  68.402973
1   Q    C      120       3   5     2      8   top  33.983679
```

[21]:
```
num_feature = ['calories', 'protein', 'fat', 'fiber', 'sugars']
x[num_feature].head()
```

[21]:
```
0 calories protein fat fiber sugars
0      NaN       4   1    10      6
1      120       3   5     2      8
2       70       4   1     9      5
3       50       4   0    14      0
4      NaN       2   2     1      8
```

[22]:
```
# Impute null values using SImpleImputer using the ‘mean’ strategy.
impute = SimpleImputer(strategy='mean')
x[num_feature] = impute.fit_transform(x[num_feature])
```

[23]:
```
# Scale the data with StandardScaler
scaler = StandardScaler()
x[num_feature] = scaler.fit_transform(x[num_feature])
```

[24]:
```
# removing scientific notation
pd.options.display.float_format = '{:.4f}'.format

# Display the pipeline to confirm the code was error-free
x[num_feature].head()
```

[24]:
```
0  calories  protein     fat   fiber  sugars
0   -0.0000   1.3373 -0.0441  3.3727 -0.2088
1    0.7515   0.4179  4.0677 -0.0709  0.2618
2   -2.0090   1.3373 -0.0441  2.9422 -0.4440
3   -3.1132   1.3373 -1.0720  5.0945 -1.6204
4   -0.0000  -0.5015  0.9839 -0.5014  0.2618
```

0.1 Part 3: Create a Column Transformer

[25]:
```
# part 3 x3 and y3
x3 = df.drop(['rating'], axis=1)
y3 = df['rating']
```

[26]:
```
# spliting the x3 and y3
x3_train, x3_test, y3_train, y3_test = train_test_split(x3, y3, test_size=0.2,␣
 ↪random_state=42)
```

[33]:
```
# Define 3 tuples (one for each pipeline that includes the name, the pipeline␣
 ↪object, and the list of columns to apply it to.)
ordinal_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ordinal', OrdinalEncoder()),
    ('scaler', StandardScaler())])
ordinal_turple = (ordinal_pipeline, ordinal_features)

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='MISSING')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
cat_turple = (cat_pipeline, cat_features)

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
num_turple = (num_pipeline, num_feature)
```

[34]:
```
# Create one column transformer object that includes the 3 preprocessing␣
 ↪pipelines you created in the previous assignment.
preprocessor = ColumnTransformer(
    transformers=[
        ('ordinal', ordinal_pipeline, ordinal_features),
        ('categorical', cat_pipeline, cat_features),
        ('numerical', num_pipeline, num_feature)
    ]
)
```

[35]:
```
preprocessor
```

[35]:
```
ColumnTransformer(transformers=[('ordinal',
                                 Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                 ('ordinal', OrdinalEncoder()),
                                                 ('scaler', StandardScaler())]),
                                 ['shelf']),
                                ('categorical',
                                 Pipeline(steps=[('imputer',
SimpleImputer(fill_value='MISSING',
strategy='constant')),
                                                 ('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
                                 ['mfr', 'type']),
                                ('numerical',
                                 Pipeline(steps=[('imputer', SimpleImputer()),
                                                 ('scaler', StandardScaler())]),
                                 ['calories', 'protein', 'fat', 'fiber',
                                  'sugars'])])
```

[37]:
```
# Save the transformed training data as X_train_processed
X_train_processed = preprocessor.fit_transform(x3_train)
```

[42]:
```
# Get new column names (categorical features will expand due to OneHotEncoder)
cat_encoded_columns = preprocessor.named_transformers_['categorical'].
 ↪named_steps['onehot'].get_feature_names_out(cat_features)

# Combine all column names
new_column_names = ordinal_features + list(cat_encoded_columns) + num_feature
```

[43]:
```
X_train_processed[:5]

# Display the .head() of X_train_processed
X_train_processed_df = pd.DataFrame(X_train_processed, columns =␣
 ↪new_column_names)
X_train_processed_df.head()
```

[43]:
```
     shelf  mfr_A  mfr_G  mfr_K  mfr_MISSING  mfr_N  mfr_P  mfr_Q  mfr_R  \
0   0.9542 0.0000 0.0000 0.0000       0.0000 0.0000 1.0000 0.0000 0.0000
1  -1.3740 0.0000 1.0000 0.0000       0.0000 0.0000 0.0000 0.0000 0.0000
2   0.9542 0.0000 0.0000 0.0000       0.0000 0.0000 1.0000 0.0000 0.0000
3   0.9542 0.0000 1.0000 0.0000       0.0000 0.0000 0.0000 0.0000 0.0000
4  -1.3740 0.0000 0.0000 0.0000       0.0000 0.0000 1.0000 0.0000 0.0000

   type_C  type_H  calories  protein     fat   fiber  sugars
0  1.0000  0.0000   -0.8264   0.3702  0.0000  0.0000  0.0000
1  1.0000  0.0000    0.1818  -0.5331  1.0155 -0.2419  0.8267
2  1.0000  0.0000    0.6859   0.3702  2.0502  0.4189 -0.5942
3  1.0000  0.0000   -0.3223  -0.5331  0.0000 -0.0216  0.8267
4  1.0000  0.0000   -0.3223  -0.5331 -1.0538 -0.9026  2.0107
```

[39]:
```
# Save the transformed testing data as X_ test_processed
X_test_processed = preprocessor.transform(x3_test)
```

[44]:
```
X_test_processed[:5]

# Display the .head() of the X_test_processed
X_test_processed_df = pd.DataFrame(X_test_processed, columns = new_column_names)
X_test_processed_df.head()
```

[44]:
```
    shelf  mfr_A  mfr_G  mfr_K  mfr_MISSING  mfr_N  mfr_P  mfr_Q  mfr_R  \
0  0.9542 0.0000 0.0000 0.0000       0.0000 0.0000 0.0000 0.0000 1.0000
1  0.9542 0.0000 0.0000 0.0000       0.0000 0.0000 0.0000 1.0000 0.0000
2 -0.2099 0.0000 0.0000 0.0000       0.0000 0.0000 0.0000 1.0000 0.0000
3  0.9542 0.0000 0.0000 0.0000       0.0000 1.0000 0.0000 0.0000 0.0000
4  0.9542 0.0000 0.0000 0.0000       0.0000 0.0000 0.0000 0.0000 1.0000

   type_C  type_H  calories  protein     fat   fiber  sugars
0  1.0000  0.0000    0.0000  -0.5331  1.0155 -0.4621  0.3531
1  1.0000  0.0000    0.0000  -1.4364  1.0155 -0.4621  1.0635
2  1.0000  0.0000    0.0000  -1.4364  1.0155 -0.9026  1.3003
3  1.0000  0.0000    0.0000   1.2736 -0.0192  3.5024 -0.1206
4  1.0000  0.0000    0.0000   1.2736  2.0502  0.4189  1.0635
```

[ ]: