Model

December 3, 2023

[1]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
import pandas as pd
```

[2]:
```
# Read data from Excel file into a Pandas DataFrame
file_path = 'dm_mimic_pathways.csv'
df = pd.read_csv(file_path)
```

[3]:
```
column_name_mapping = {'person_id': 'Person',
                       'race_concept_id': 'Race',
                       'gender_concept_id':'Gender',
                       'age_group':'Age Group',
                       'pathways':'Treatment Regimen'}

race_mapping = {8527: 'White/ Hispanic',
                8516: 'Black',
                8515: 'Asian',
                0:'Unknown',
                38003592: 'Asian',
                4077359: 'Other',
                4218674: 'Unknown',
                4188159: 'White/ Hispanic',
                38003599: 'Black',
                38003574: 'Asian',
                4212311: 'Asian',
                38003600: 'Black',
                8557: 'Other',
                38003584: 'Asian',
                38003578: 'Asian',
                4087921: 'Other',
                38003615: 'Other',
                38003581: 'Asian',
                8657: 'Other',
                38003579: 'Asian',
```

```
                38003605: 'Black',
                38003614: 'White',
                4213463: 'White'}

gender_mapping = {8507: 'Male',
                  8532: 'Female'}

age_mapping = {'10 - 19': 'Teens',
               '20 - 29': 'Twenties',
               '30 - 39': 'Thirties',
               '40 - 49': 'Forties',
               '50 - 59': 'Fifties',
               '60 - 69': 'Sixties',
               '70 - 79': 'Seventies',
               '80 - 89': 'Eighties',
              '> 90': 'Nineties'}
```

[4]:
```
df = df.rename(columns=column_name_mapping)
df['Race'] = df['Race'].replace(race_mapping)
df['Gender'] = df['Gender'].replace(gender_mapping)
df['Age Group'] = df['Age Group'].replace(age_mapping)
df['Age Group'].fillna('Unknown', inplace=True)
```

[5]:
```
df = df[(df['Age Group'] != 'Unknown') & (df['Race'] != 'Unknown')]
```

[6]:
```
print(len(df))
n = 1
values_to_preserve = df['Treatment Regimen'].value_counts().head(n)
print(values_to_preserve)
```

```
1746
Treatment Regimen
19071700    463
Name: count, dtype: int64
```

[7]:
```
def preserve_or_change(value, value_set, replacement_value):
    return value if value in value_set else replacement_value
```

[8]:
```
df['Treatment Regimen'] = df['Treatment Regimen'].apply(lambda x:␣
 ↪preserve_or_change(x, values_to_preserve, 'Other'))
df.head(5)
len(df['Treatment Regimen'].unique())
```

[8]:
```
2
```

[9]:
```
X = df[['Age Group', 'Race', 'Gender']]
y = df['Treatment Regimen']
```

[10]:
```
preprocessor = ColumnTransformer(
    transformers=[
            ('cat', OneHotEncoder(), ['Age Group', 'Race', 'Gender'])
        ],
        remainder='passthrough'

)
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(multi_class='multinomial', class_weight =␣
 ↪'balanced'))
])
```

[11]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[12]:
```
# Train the model
pipeline.fit(X_train, y_train)
```

[12]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat', OneHotEncoder(),
                                                  ['Age Group', 'Race',
                                                   'Gender'])])),
                ('classifier',
                 LogisticRegression(class_weight='balanced',
                                    multi_class='multinomial'))])
```

[13]:
```
# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Create a DataFrame with actual and predicted values
df_predictions = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred
})

print("Actual vs Predicted:")
print(df_predictions)
```

```
Accuracy: 0.55
Actual vs Predicted:
        Actual Predicted
```

```
408   19071700     Other
387   19071700  19071700
803   19071700  19071700
81       Other  19071700
942   19071700     Other
…          …        …
596      Other  19071700
1710     Other     Other
894      Other     Other
1226     Other     Other
1466     Other  19071700

[350 rows x 2 columns]
```

[14]:
```
# Access the one-hot encoder from the pipeline
encoder = pipeline.named_steps['preprocessor'].named_transformers_['cat']

# Get feature names after one-hot encoding
feature_names_after_encoding = list(encoder.get_feature_names_out(X.
 ↪select_dtypes(include=['object']).columns))

# Concatenate feature names with numeric features
all_feature_names = X.select_dtypes(include=['number']).columns.tolist() +␣
 ↪feature_names_after_encoding

# Access the model from the pipeline
model = pipeline.named_steps['classifier']

# Get coefficients
coefficients = model.coef_

# Display coefficients in a DataFrame
df_coefficients = pd.DataFrame(coefficients, columns=all_feature_names)
df_coefficients['Intercept'] = model.intercept_
df_coefficients['Class'] = model.classes_
df_coefficients.set_index('Class', inplace=True)

print("Coefficients:")
print(df_coefficients)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[14], line 19
     17 df_coefficients = pd.DataFrame(coefficients, columns=all_feature_names)
     18 df_coefficients['Intercept'] = model.intercept_
---> 19 df_coefficients['Class'] = model.classes_
     20 df_coefficients.set_index('Class', inplace=True)
```

```
     22 print("Coefficients:")

File ~/Programs/Miniconda3/envs/datsci/lib/python3.10/site-packages/pandas/core
 ↪frame.py:4094, in DataFrame.__setitem__(self, key, value)
   4091     self._setitem_array([key], value)
   4092 else:
   4093     # set column
-> 4094     self._set_item(key, value)

File ~/Programs/Miniconda3/envs/datsci/lib/python3.10/site-packages/pandas/core
 ↪frame.py:4303, in DataFrame._set_item(self, key, value)
   4293 def _set_item(self, key, value) -> None:
   4294     """
   4295     Add series to DataFrame in specified column.
   4296
   (…)
   4301     ensure homogeneity.
   4302     """
-> 4303     value, refs = self._sanitize_column(value)
   4305     if (
   4306         key in self.columns
   4307         and value.ndim == 1
   4308         and not isinstance(value.dtype, ExtensionDtype)
   4309     ):
   4310         # broadcast across multiple columns if necessary
   4311         if not self.columns.is_unique or isinstance(self.columns,␣
 ↪MultiIndex):

File ~/Programs/Miniconda3/envs/datsci/lib/python3.10/site-packages/pandas/core
 ↪frame.py:5042, in DataFrame._sanitize_column(self, value)
   5039     return _reindex_for_setitem(value, self.index)
   5041 if is_list_like(value):
-> 5042     com.require_length_match(value, self.index)
   5043 return sanitize_array(value, self.index, copy=True, allow_2d=True), Non

File ~/Programs/Miniconda3/envs/datsci/lib/python3.10/site-packages/pandas/core
 ↪common.py:561, in require_length_match(data, index)
    557 """
    558 Check the length of data matches the length of the index.
    559 """
    560 if len(data) != len(index):
--> 561     raise ValueError(
    562         "Length of values "
    563         f"({len(data)}) "
    564         "does not match length of index "
    565         f"({len(index)})"
    566     )
```

```
ValueError: Length of values (2) does not match length of index (1)
```