PDP Pipeline homework

Naam: Ruben Stoop
Studentnummer: 670240

Introduction

In this document you will find the assignments for PDP. The git repository for these assignments can be found at: https://github.com/neburpoots/PDPHomeworkPipeline
This document will contain screenshots of some of the results but it's best to look at the entire Jupyter notebook file.

Feature extraction

The first thing I wanted to do is check the dataset for correlations. My way of doing this was to change all classifications with numbers and check the correlation with a heatmap. Figure 1 is the result of this. As you can see the Cumulative deaths, Cumulative cases and New cases have a high correlation with New_deaths. All of the X variables that were classes like date, Country and WHO_region have a low correlation. **I Have removed the Country_code from the dataset since this will be the same as the country.**

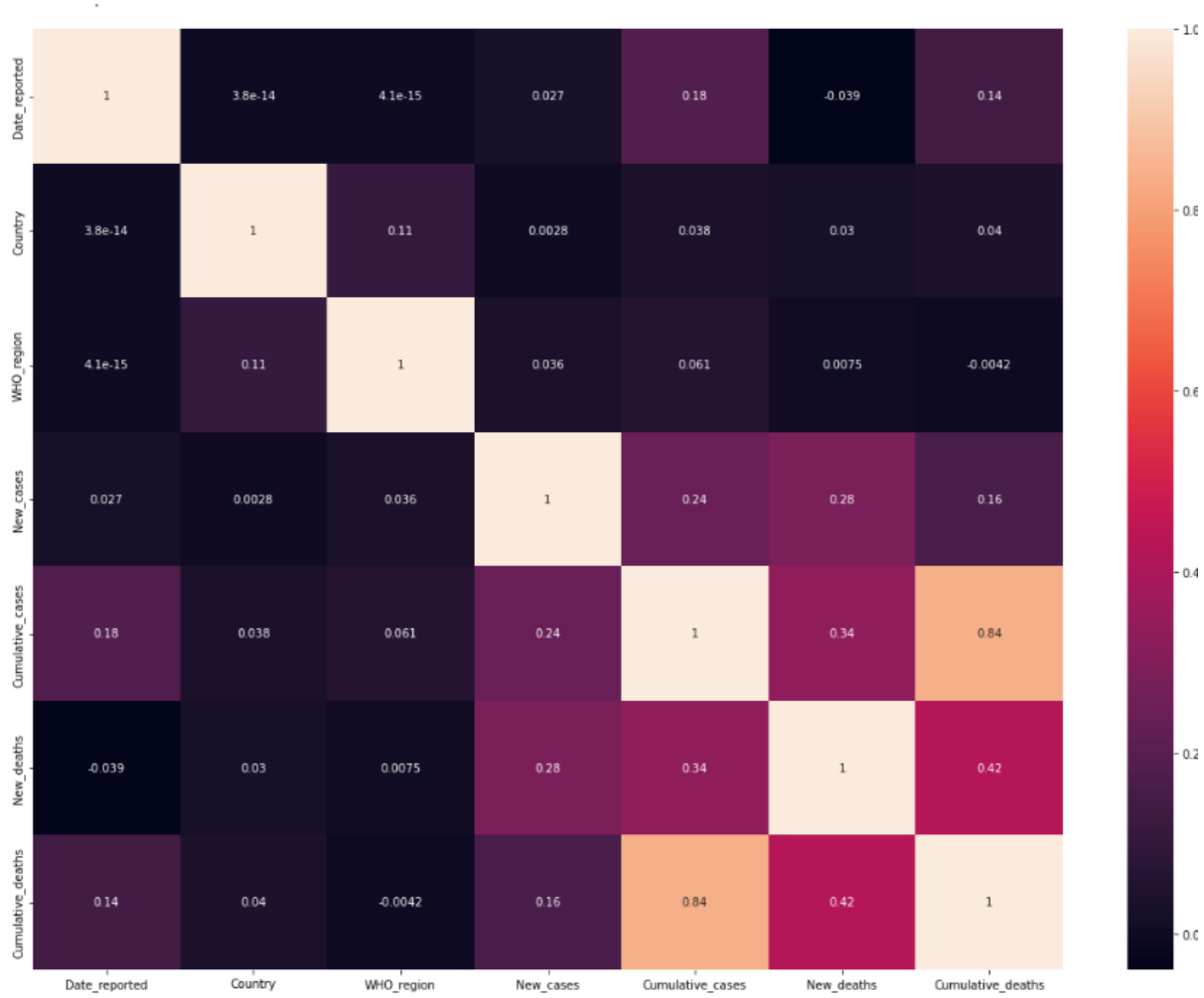

Figure 1. Heatmap with the correlations of all variables.

After this I decided to make three more feature extraction methods. I copied these from my Peter Stikker Regression homework and adapted these accordingly.

The result of the second feature extraction method using statsmodel results in the following where all of the X variables are compared with the Dep. Variable New_deaths. The importance here is the coefficient, P value and the Adj. R-squared for the total.

[6]:

OLS Regression Results

Dep. Variable:	New_deaths	**R-squared:**	0.234
Model:	OLS	**Adj. R-squared:**	0.234
Method:	Least Squares	**F-statistic:**	8688.
Date:	Sat, 20 May 2023	**Prob (F-statistic):**	0.00
Time:	13:40:56	**Log-Likelihood:**	-1.0742e+06
No. Observations:	171066	**AIC:**	2.148e+06
Df Residuals:	171059	**BIC:**	2.149e+06
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	28.4821	0.886	32.163	0.000	26.746	30.218
x1	-0.0402	0.001	-43.920	0.000	-0.042	-0.038
x2	0.0260	0.005	5.661	0.000	0.017	0.035
x3	0.0009	8.27e-06	104.908	0.000	0.001	0.001
x4	-2.647e-06	1.02e-07	-25.934	0.000	-2.85e-06	-2.45e-06
x5	0.5526	0.158	3.504	0.000	0.243	0.862
x6	0.0010	8.19e-06	122.751	0.000	0.001	0.001

Omnibus:	327945.786	**Durbin-Watson:**	2.008
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3105734738.936
Skew:	14.336	**Prob(JB):**	0.00
Kurtosis:	662.471	**Cond. No.**	1.68e+07

Figure 2. Statsmodel api result

The third feature extraction method is calculating the VIF score for each X variable. VIF score is the strength of the correlation between independent variables.

Feature extraction method 3

```
]: # features = convert_dataset_to_numbers()
features = convert_dataset_to_numbers()
features = sm.add_constant(features)

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = features.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(features.values, i)
                          for i in range(features.shape[1])]

print(vif_data)
```

```
              feature       VIF
0               const  8.094438
1       Date_reported  1.047727
2             Country  1.013868
3          WHO_region  1.027013
4           New_cases  1.138427
5    Cumulative_cases  3.655421
6          New_deaths  1.316235
7   Cumulative_deaths  3.798505
```

Figure 3. VIF scores of all variables

The fourth feature extraction method is checking all combinations to get the highest possible r-value. Figure 4 is an example of this.

Feature extraction method 4

```
# all possible columns for X
# define Y as same accross the loops

covid_cases = convert_dataset_to_numbers()

cols = [x for x in covid_cases.columns if x != 'New_deaths']

# define result dictionary
fit_d = {}

# loop for any length of combinations
for i in range(1, len(cols)+1):
    # loop for any combinations with length i
    for comb in combinations(cols, i):


        # Define X from the combination
        X = covid_cases[list(comb)]
        X = sm.add_constant(X)

        # perform the OLS opertion
        model = sm.OLS(y,X, missing = 'drop').fit()
        # save the rsquared in a dictionnary
        fit_d[comb] = model.rsquared

# extract the key for the max R value
key_max = max(fit_d, key=fit_d.get)

print(f'Best variables {key_max} for a R-value of {round(fit_d[key_max], 5)}')
```

```
Best variables ('Date_reported', 'Country', 'WHO_region', 'New_cases', 'Cumulative_cases', 'Cumulative_deaths') f
a R-value of 0.24026
```

Based on the feature selection I have decided to use all of the variables

Figure 4. Code snippet of checking all combinations for highest r-value.

Data used for predictions

I had a lot of trouble understanding what actually needs to be predicted in the assignment. The assignment said to predict the deaths per country 1 week from each data point. The trouble with the assignment is that just taking the date plus 7 for all rows does not work since all of the other x variables will become invalid. The only variables not reliant on the date are the country, who_region and the date itself which has no correlation to New_deaths. To basically get the predictions for next week I have decided to make the test the same as the train set. Since if you want the predictions for the next week this will be possible with the same set.

I have started with an example outside of a pipeline to get a basic idea of the accuracy. Figure 5 shows the code for this.

```
train_set = convert_dataset_to_numbers()
test_set = convert_dataset_to_numbers()

train_set_x = train_set[[*key_max]]
train_set_y = train_set[['New_deaths']]

test_set_x = test_set[[*key_max]]
test_set_y = test_set[['New_deaths']]

standard_scaler = StandardScaler()
train_set_x = standard_scaler.fit_transform(train_set_x)
test_set_x = standard_scaler.transform(test_set_x)

hidden_units1 = 160
hidden_units2 = 480
hidden_units3 = 156

model = Sequential([
    Dense(hidden_units1, kernel_initializer='normal', activation='relu'),
    Dense(hidden_units2, kernel_initializer='normal', activation='relu'),
    Dense(hidden_units3, kernel_initializer='normal', activation='relu'),
    Dense(1, kernel_initializer='normal', activation='linear')])

msle = MeanSquaredLogarithmicError()
model.compile(
    loss=msle,
    optimizer='adam',
    metrics=[msle])


# train the model
history = model.fit(
    train_set_x,
    train_set_y,
    epochs=40,
    batch_size=64,
    validation_split=0.2
)
model.summary()
```

Figure 5. code snippet standard model.

The result of this model is figure 5.

```
Epoch 40/40
3564/3564 [==============================] - 7s 2ms/step - loss: 0.1780 - mean_squared_logarithmic_error: 0.1780 -
val_loss: 0.7485 - val_mean_squared_logarithmic_error: 0.7485
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 160)               1120

 dense_1 (Dense)             (None, 480)               77280

 dense_2 (Dense)             (None, 156)               75036

 dense_3 (Dense)             (None, 1)                 157

=================================================================
Total params: 153,593
Trainable params: 153,593
Non-trainable params: 0
_________________________________________________________________
```

Showing the result of the first 100 predictions

```
Pred_target = model.predict(test_set_x)

import numpy as np
from matplotlib.pyplot import figure

figure(figsize=(18, 6), dpi=80)

plt.plot(np.array(test_set_y)[0:1000,],  color='r', label='Actual')
plt.plot(Pred_target[0:1000,], color='b', label='Predited')
# Adding legend, which helps us recognize the curve according to it's color
plt.legend()

# To load the display window
plt.show()
```

```
8910/8910 [==============================] - 9s 965us/step
```

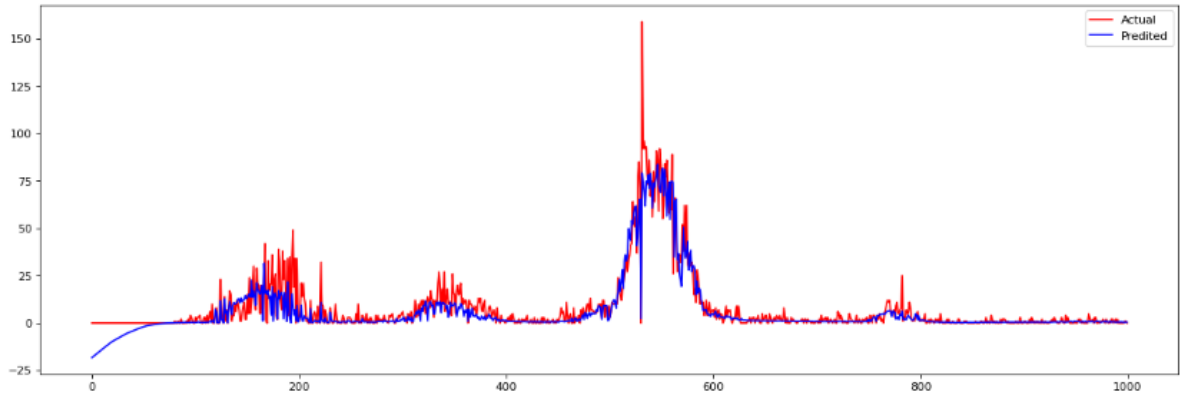

Figure 5. Result of model without pipeline

After this I tried a basic example of a pipeline. I had the most difficulty with preprocessing the data since the dates and countries had to be converted to numbers.

First example of a pipeline with a LinearRegression model

```
# Define the features and target columns
features = [*key_max]
target = 'New_deaths'
class_columns = ['Country', 'WHO_region']

data = pd.read_csv('WHO-COVID-19-global-data.csv')


# Split the dataframe into features and target
X = data[features]
y = data[target]

# Create a ColumnTransformer to handle data preprocessing for different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['New_cases', 'Cumulative_cases', 'Cumulative_deaths']),  # Numerical features: n
        ('date', OrdinalEncoder(dtype=int, handle_unknown='use_encoded_value', unknown_value=-1), ['Date_reported']
        ('cat', OneHotEncoder(), class_columns)  # Class column: one-hot encode
    ])

# Define the regression model
model = LinearRegression()

# Create the pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', model)
])


# Fit the pipeline to the data
pipeline.fit(X, y)


# Use the pipeline to make predictions
predictions = pipeline.predict(X)

# Evaluate the pipeline on the test data
score = pipeline.score(X, y)

print("Model score: %.3f" % score)

import numpy as np
from matplotlib.pyplot import figure

figure(figsize=(18, 6), dpi=80)

plt.plot(np.array(y)[0:1000,],  color='r', label='Actual')
plt.plot(predictions[0:1000,], color='b', label='Predited')
# Adding legend, which helps us recognize the curve according to it's color
plt.legend()

# To load the display window
plt.show()
```

Figure 6. Snippet of code of first pipeline

The result of this model is shown in Figure 7. As you can see it's not exactly what you would call accurate.

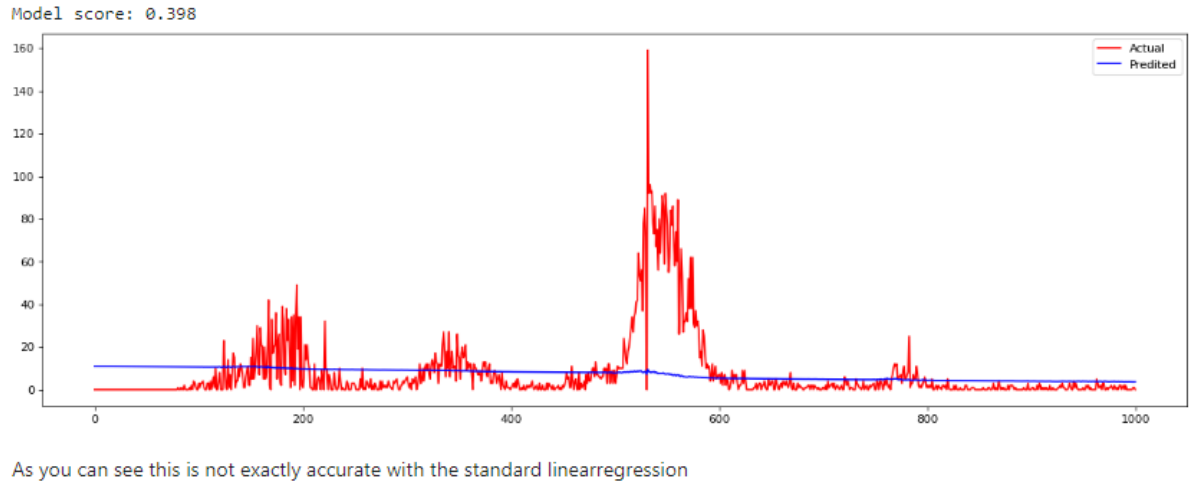

Figure 7. Result of this LinearRegression() model

My plan now was to implement the model I have previously made into the pipeline. This however was immensely difficult. I had spent a total of 8 hours trying to get it to work. The problem was that the categorical data could for some reason not be properly transformed into data that the model could use to train. I've tried all types of different custom models with different ways of data conversion with sparse, csr_matrix, numpy array and a dataframe but it would not work. So as a inbetween solution I have refactored the X variables that are classes to numbers before putting them in the pipeline. Below is the final result.

Pipeline with custom model

```
[*]: hidden_units1 = 160
hidden_units2 = 480
hidden_units3 = 156

# Load the dataset
data=convert_dataset_to_numbers()

# Define the features and target column
features = data.drop('New_deaths', axis=1)

target = data['New_deaths']

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Create the preprocessor for numerical features
numeric_features = ['Cumulative_cases', 'New_cases', 'Cumulative_deaths', 'Date_reported', 'Country', 'WHO_region']
numeric_transformer = StandardScaler()

# Create the preprocessor for categorical features
categorical_features = []
categorical_transformer = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(), categorical_features)
    ],
    remainder='passthrough'
)
```

Figure 8. Code snippet part 1

```
# Combine the preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('numeric', numeric_transformer, numeric_features),
    ], remainder='passthrough')

# Custom transformer to convert DataFrame to numpy array
class DataFrameToArrayTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return pd.DataFrame(data=X)

# Define the model architecture
def create_model():
    model = Sequential([
                Dense(hidden_units1, kernel_initializer='normal', activation='relu'),
                Dense(hidden_units2, kernel_initializer='normal', activation='relu'),
                Dense(hidden_units3, kernel_initializer='normal', activation='relu'),
                Dense(1, kernel_initializer='normal', activation='linear')])

    msle = MeanSquaredLogarithmicError()
    model.compile(loss=msle,
                optimizer='adam',
                metrics=[msle])
    return model

# Create the KerasRegressor with the custom model
regressor = KerasRegressor(build_fn=create_model, epochs=40, batch_size=32)

# Create the pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('dataframe_to_array', DataFrameToArrayTransformer()),  # Convert DataFrame to numpy array
    ('regressor', regressor)
])
# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)

# Evaluate the pipeline on the test data
score = pipeline.score(X_test, y_test)
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print('Mean Squared Error:', mse)
print('R-squared:', r2)
print("Model score: %.3f" % score)
```

Figure 9. Code snippet part 2.

The code snippet produces the following result.

```
7128/7128 [==============================] - 11s 2ms/step - loss: 0.1805 - mean_squared_logarithmic_error: 0.1805
Epoch 36/40
7128/7128 [==============================] - 11s 2ms/step - loss: 0.1798 - mean_squared_logarithmic_error: 0.1798
Epoch 37/40
7128/7128 [==============================] - 11s 2ms/step - loss: 0.1796 - mean_squared_logarithmic_error: 0.1796
Epoch 38/40
7128/7128 [==============================] - 11s 2ms/step - loss: 0.1787 - mean_squared_logarithmic_error: 0.1787
Epoch 39/40
7128/7128 [==============================] - 11s 2ms/step - loss: 0.1759 - mean_squared_logarithmic_error: 0.1759
Epoch 40/40
7128/7128 [==============================] - 11s 2ms/step - loss: 0.1773 - mean_squared_logarithmic_error: 0.1773
1782/1782 [==============================] - 2s 1ms/step - loss: 0.1859 - mean_squared_logarithmic_error: 0.1859
Mean Squared Error: 3784.2234948008704
R-squared: 0.79861894674696
Model score: -0.186
```

Showing results in a graph

```
]: # Use the pipeline to make predictions
predictions = pipeline.predict(X_test)

figure(figsize=(18, 6), dpi=80)

plt.plot(np.array(y_test)[0:100,],  color='r', label='Actual')
plt.plot(predictions[0:100,], color='b', label='Predited')
# Adding legend, which helps us recognize the curve according to it's color
plt.legend()

# To load the display window
plt.show()
```

```
1782/1782 [==============================] - 2s 972us/step
```

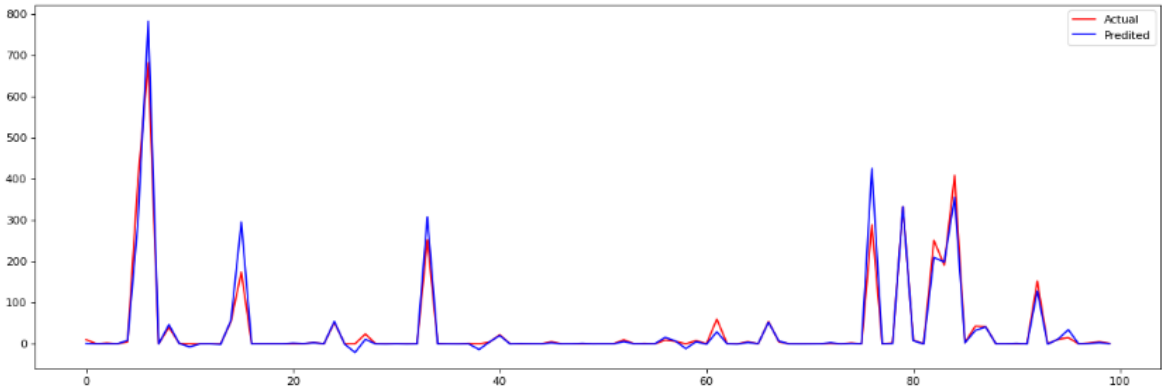

Figure 10. Result of the pipeline with custom model