baseline_model

January 2, 2024

1 Baseline Model

The goal of this notebook is to develop a scrappy baseline model which we can improve on later if we decide to implement an LSTM or otherwise. But beforehand, we need to process the data and determine what input features to use for the ANN.

1.1 Pre-processing

1. Construct the feature/target set.
2. Scale the data
3. Split train/test

[1]:
```
import pandas as pd
import sys
import os
import datetime
import numpy as np
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt
import plotly.express as px
import numpy as np
import seaborn as sns
sys.path.insert(0, os.path.abspath('../'))

from lib.common import constants as k
from lib.eirgrid import data as eirgrid_data
from lib.common.marginal_emissions import compute_mef
from lib.common.data_window import DataWindow

# Eirgrid system
eirgrid = eirgrid_data.system()
eirgrid = compute_mef(eirgrid).dropna()
```

There are some instances where the MEF is infinite. This is because $\Delta PG_{i,t+1} = 0$ meaning there was no change in generation

The simplest thing to do is backfillt this with the MEF of the previous time step. The assumption being if there was no change in generation, the MEF should remain the same.

[2]:
```
print('There are %d inf MEF values' % eirgrid['MarginalEmissions'].isin([np.
    ↪inf, -np.inf]).sum())
eirgrid = eirgrid.replace([np.inf, -np.inf], np.nan)
eirgrid = eirgrid.ffill()
print('There are %d inf MEF values' % eirgrid['MarginalEmissions'].isin([np.
    ↪inf, -np.inf]).sum())
```

```
There are 731 inf MEF values
There are 0 inf MEF values
```

[3]:
```
eirgrid.corr(numeric_only=True)['MarginalEmissions']
```

[3]:
```
SysFrequency          0.003779
Co2Emissions          0.020952
Co2Intensity          0.024599
SystemDemand          0.005024
GenExp               -0.004069
InterNet              0.004067
WindActual           -0.025451
MarginalEmissions     1.000000
Name: MarginalEmissions, dtype: float64
```

Linear correlation methods are unable to capture the relationship of the input features given the lack of normalality and non-linearity. It appears that the features are uncorrelated as most of the r values are near zero.

A way to overcome this is through the application of Grey Relational Analysis (GRA) which can describe the relationship between these input features more aptly.

1.2 Feature Selection

Use a combination of Grey relational analysis and PCA to figure out which features are most important. This requires that the data is scaled and the timestamps are properly encoded.

- To encode time stamps create two features to represent the cosine and sine of the time.
- Use a `MinMaxScaler` to scale the data.

This method was employed by researchers building an LSTM ANN to forecast CO_2 emissions in China [1] - Domain knowledge and literature review was used to determine a list of 16 candidate features - The key influencing factors are obtained using Grey Relational analysis - Principal component analysis is performed on these factors to obtain the main information affecting carbon emissions & simplify input variables

---

[1] Y. Huang, L. Shen, and H. Liu, 'Grey relational analysis, principal component analysis and forecasting of carbon emissions based on long short-term memory in China', Journal of Cleaner Production, vol. 209, pp. 415–423, Feb. 2019, doi: 10.1016/j.jclepro.2018.10.128.

[4]:

```python
def feature_engineer(frame: pd.DataFrame, n_lag=0, target_column='MarginalEmissions', datetime_column='EffectiveTime') -> pd.DataFrame:
    df = frame.copy()

    # Apply a sine/cosine transformation to the timestamp to preserve the cyclical nature of the day
    # In a way that can be interpreted by the ML
    timestamp_s = df['EffectiveTime'].map(datetime.datetime.timestamp)
    day_s = 24 * 60 * 60
    features = df.drop(columns=['EffectiveTime'])

    feature_columns = features.columns
    # Lag input columns
    for lag in range(n_lag):
        for column in feature_columns:
            if column == target_column:
                continue
            features[f'{column}_lag_{lag+1}'] = features[column].shift(-1 * lag)

    features['day_sin'] = (np.sin(timestamp_s * (2*np.pi/day_s))).values
    features['day_cos'] = (np.cos(timestamp_s * (2*np.pi/day_s))).values

    # Remove the undefined features
    features = features.dropna()

    # Scale the data
    scaler = MinMaxScaler()
    scaler.fit(features)
    features[features.columns] = scaler.transform(features[features.columns])
    target = features[target_column]
    features = features.drop(columns=target_column)
    return (target, features)

target, features = feature_engineer(eirgrid)
display(features)
```

	SysFrequency	Co2Emissions	Co2Intensity	SystemDemand	GenExp	\
56	0.511628	0.009233	0.075472	0.468589	0.572204	
57	0.534884	0.014915	0.086792	0.450034	0.555917	
58	0.604651	0.013139	0.090566	0.431701	0.554651	
59	0.581395	0.006037	0.086792	0.412028	0.551393	
60	0.441860	0.011364	0.100000	0.391907	0.522439	
…	…	…	…	…	…	

```
122476      0.697674      0.303977      0.581132      0.158059  0.254434
122477      0.767442      0.337358      0.620755      0.168791  0.260043
122478      0.720930      0.372159      0.635849      0.186452  0.288455
122479      0.604651      0.389560      0.630189      0.203666  0.313246
122480      0.651163      0.389205      0.633962      0.216186  0.309989

        InterNet  WindActual   day_sin   day_cos
56      0.387210    0.822739  0.804381  0.103323
57      0.399295    0.801195  0.829673  0.124080
58      0.361531    0.805887  0.853553  0.146447
59      0.118832    0.806741  0.875920  0.170327
60      0.360524    0.761519  0.896677  0.195619
…          …          …        …         …
122476  0.446123    0.065060  0.500000  1.000000
122477  0.455186    0.061860  0.467298  0.998929
122478  0.506546    0.063567  0.434737  0.995722
122479  0.388218    0.064420  0.402455  0.990393
122480  0.588117    0.065913  0.370590  0.982963

[119997 rows x 9 columns]
```

1.2.1 Grey Relational Analysis

We're dealing with a multivariate time series (more than one input). This makes it hard to determine the relationship between our input features and targets.

As a part of the network design we want to use as little input features as possible while still capturing the underlying relationships in the data. We also want input features that are easy to derive to make predictions easier.

Greys Relational Analysis is a method to understand this further.

> GRA is employed to search for Grey Relational Grade (GRG) which can be used to describe the relationships between the data attributes and to determine the important factors that significantly influence some defined objectives. [2]

Methodology Preprocessing

- The nescessary preprocessing has been done (scaling, normalising)
- The original data series X is split into a reference series x_0 and comparative series x_i. In this example, x_0 is the target feature (.i.e. MEF) and x_i represent the input factors

Grey Relational Coefficent

$$\zeta_i(k) = \frac{\Delta\min + \zeta\Delta\max}{\Delta_{0,j}(k) + \zeta\Delta\max}$$

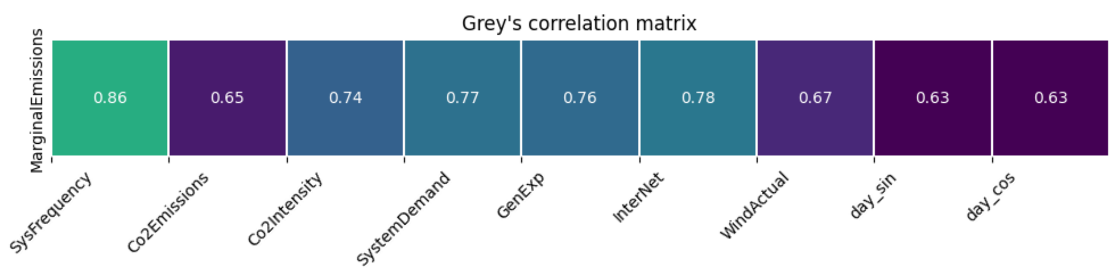
---

[2] R. Sallehuddin, S. M. Hj. Shamsuddin, and S. Z. M. Hashim, 'Application of Grey Relational Analysis for Multivariate Time Series', in 2008 Eighth International Conference on Intelligent Systems Design and Applications, Nov. 2008, pp. 432–437. doi: 10.1109/ISDA.2008.181.

[5]:
```
def grey_relational_grade(features: pd.DataFrame, target: pd.Series, zeta=0.5,␣
↪norm=False) -> pd.DataFrame:
    """
    Returns the grey relational grade for a feature set and target data frame.
    Data should be scaled before processing.
    """
    # Convert the DataFrame and Series to numpy arrays
    feature_data = features.values
    target_data  = target.values.reshape(-1, 1)  # Reshape target to be a 2D␣
↪array for concatenation

    # Combine the target and features into one array with the target as the␣
↪first column
    data = np.concatenate([target_data, feature_data], axis=1)

    # Normalize the data using MinMaxScaler
    if norm:
        scaler = MinMaxScaler()
        data = scaler.fit_transform(data)

    # Calculate the absolute differences
    abs_diff = np.abs(data - data[:, [0]])

    # Find the minimum and maximum of the absolute differences
    min_diff = np.nanmin(abs_diff)
    max_diff = np.nanmax(abs_diff)

    # Calculation of the grey relational coefficient matrix
    grc = (min_diff + (zeta * max_diff)) / (abs_diff + (zeta * max_diff))

    # Since the first column is the target, we ignore it in the result
    grc_target = grc[:, 1:]

    # Return as data frame
    return pd.DataFrame(grc_target, columns=features.columns)

def grg_interpret(grg: pd.DataFrame, features: pd.DataFrame, target: pd.Series):
    columns = features.columns.tolist()

    plt.figure(figsize=(12,7))
    sns.heatmap(grg.mean().values.reshape(1,-1), square=True, annot=True, ␣
↪cbar=False,
                vmax=1.0, linewidths=0.1,cmap='viridis')
    plt.ylabel(target.name)
    plt.yticks([])
    plt.title("Grey's correlation matrix")
    plt.xticks(list(range(len(columns))), columns, rotation=45)
```

```
    plt.show()

    print("Ranked feature importance")
    display(grg.mean().sort_values(ascending=False))

def grg_relevant_features(grg: pd.DataFrame, threshold = 0.7):
    return grg.columns[grg.mean().ge(threshold)].tolist()
```

[6]:
```
grg = grey_relational_grade(features, target)
grg_interpret(grg, features, target)
grg_relevant_features(grg)
```

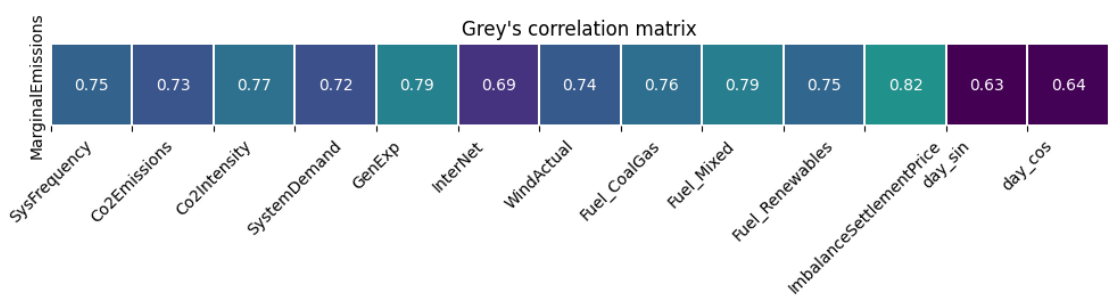



```
Ranked feature importance

SysFrequency    0.859813
InterNet        0.776019
SystemDemand    0.765004
GenExp          0.755301
Co2Intensity    0.742442
WindActual      0.669707
Co2Emissions    0.654884
day_cos         0.629277
day_sin         0.628644
dtype: float64
```

[6]: ['SysFrequency', 'Co2Intensity', 'SystemDemand', 'GenExp', 'InterNet']

[7]:
```
# Fuel mix
fuel_mix = pd.read_csv(k.PROCESSED_DATA_DIR / 'fuel_mix.csv', index_col=0)
fuel_mix.rename(columns={'StartTime': 'EffectiveTime'}, inplace=True)
fuel_mix['EffectiveTime'] = pd.to_datetime(fuel_mix['EffectiveTime'])

# Pricing data
pricing = pd.read_csv(k.RAW_DATA_DIR / 'semo' / 'price_all.csv', index_col=0)
pricing = pricing[['StartTime', 'NetImbalanceVolume',␣
 ↪'ImbalanceSettlementPrice']]
pricing.rename(columns={'StartTime': 'EffectiveTime'}, inplace=True)
```

```python
pricing['EffectiveTime'] = pd.to_datetime(pricing['EffectiveTime'])

# Reformat data frame
df = eirgrid.copy().set_index('EffectiveTime')
df = df.resample('30T').asfreq()
df = df.reset_index()

# Join with fuel mix data
df = df.merge(fuel_mix, on='EffectiveTime', how='left')
df = df.dropna()

# Join with pricing data
df = df.merge(pricing.drop(columns='NetImbalanceVolume'), on='EffectiveTime',␣
 ↪how='left')
df = df.dropna()

# Calculate MEF
target_updated, features_updated = feature_engineer(df)
grg_updated = grey_relational_grade(features_updated, target_updated)
grg_interpret(grg_updated, features_updated, target_updated)

print('Updated relevant feature set')
display(grg_relevant_features(grg_updated))
print('Complete relevant feature set')
relevant_features = set(grg_relevant_features(grg_updated) +␣
 ↪grg_relevant_features(grg))
relevant_features
```




```
Ranked feature importance

ImbalanceSettlementPrice    0.817654
GenExp                      0.793046
Fuel_Mixed                  0.788682
Co2Intensity                0.767748
Fuel_CoalGas                0.763589
Fuel_Renewables             0.752705
```

```
SysFrequency                0.748829
WindActual                  0.735526
Co2Emissions                0.728774
SystemDemand                0.723169
InterNet                    0.686831
day_cos                     0.635114
day_sin                     0.632933
dtype: float64
```

```
Updated relevant feature set
```

```
['SysFrequency',
 'Co2Emissions',
 'Co2Intensity',
 'SystemDemand',
 'GenExp',
 'WindActual',
 'Fuel_CoalGas',
 'Fuel_Mixed',
 'Fuel_Renewables',
 'ImbalanceSettlementPrice']
```

```
Complete relevant feature set
```

[7]:
```
{'Co2Emissions',
 'Co2Intensity',
 'Fuel_CoalGas',
 'Fuel_Mixed',
 'Fuel_Renewables',
 'GenExp',
 'ImbalanceSettlementPrice',
 'InterNet',
 'SysFrequency',
 'SystemDemand',
 'WindActual'}
```

1.3 Principal Component Analysis

We've identified 8 potential input features, let's try reduce the dimensionality using PCA. PCA can help us do this by creating linear combinations of the input features in a way that maintains the underlying variance in the dataset.

We can also use this to further identify and isolate the mitigating factors to build out the baseline model

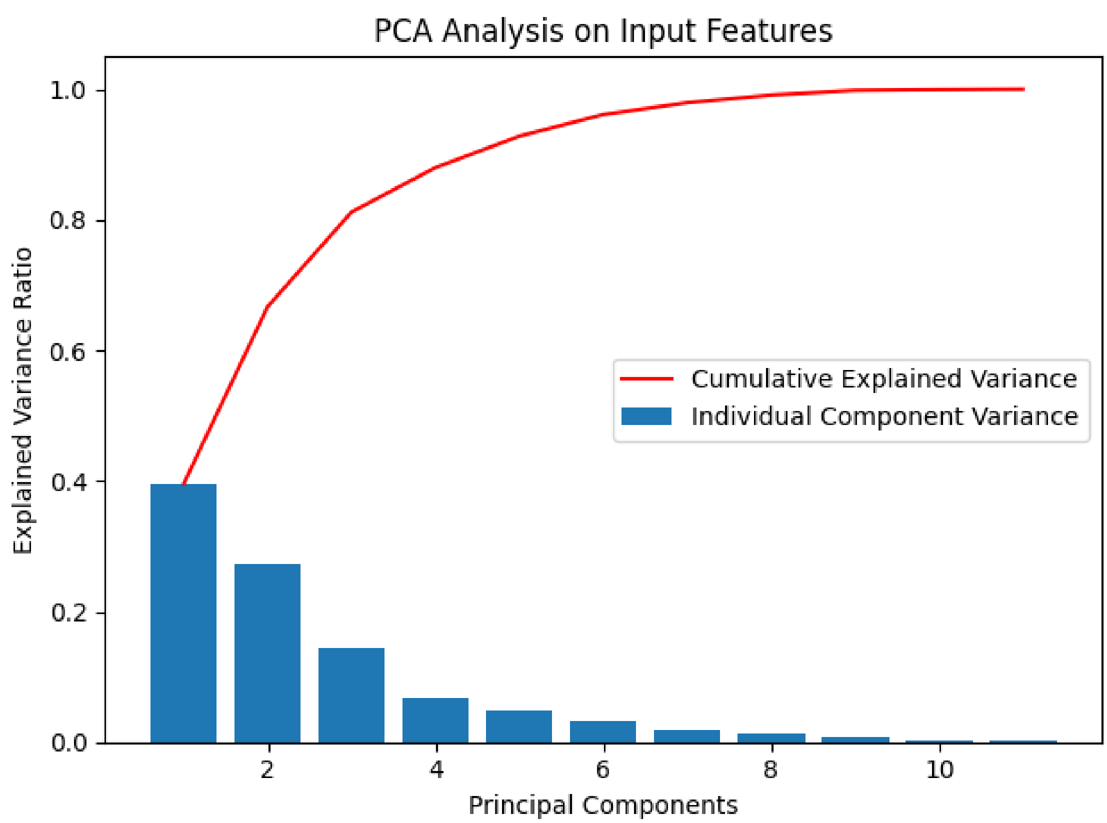
[30]:
```
features_df = features_updated[list(relevant_features)]
pca_features = features_df.values
pca_target = target_updated.values
pca = PCA(n_components=0.95)
pca_components = pca.fit_transform(pca_features)
```

```python
pca_variances = pca.explained_variance_ratio_
principal_df = pd.DataFrame(data=pca_components, columns=[f'PCA_{i}' for i in␣
  ↪range(len(pca_variances))])
principal_df['target'] = pca_target
combined_variance = np.sum(pca.explained_variance_ratio_)*100
display(principal_df)
display(pca_variances)
print('Using %d/%d components %.2f%% of the variance is retained' %␣
  ↪(len(pca_variances), len(features_df.columns), combined_variance))

# We can recover the most significant feature in each principal component
# This is given to us in `pca.components_`
n_pcs = pca.n_components_ # get number of component
# get the index of the most important feature on EACH component
most_important = [np.abs(pca.components_[i]).argmax() for i in range(n_pcs)]
initial_feature_names = features_df.columns
# get the most important feature names
most_important_features = [(i, initial_feature_names[most_important[i]], pca.
  ↪components_[i][most_important[i]]) for i in range(n_pcs)]

# print('PCA Components')
# display(pca.components_)

print('The most important features from the PCA analysis are')
most_important_cols = list(map(lambda x: x[1], most_important_features))
most_important_features
```

```
         PCA_0     PCA_1     PCA_2     PCA_3     PCA_4     PCA_5    target
0    -0.376628 -0.280247  0.063409 -0.017289  0.019142 -0.214486  0.379369
1    -0.410792 -0.316278 -0.101478  0.122934  0.003571 -0.259731  0.387421
2    -0.393784 -0.374766 -0.013671  0.057091  0.044834 -0.228053  0.379234
3    -0.381064 -0.400129 -0.104976 -0.067531  0.092531 -0.227409  0.376985
4    -0.369560 -0.425808 -0.109264 -0.105876  0.110879 -0.219335  0.378388
…          …        …        …        …        …        …        …
3721 -0.516326 -0.117343 -0.008581 -0.124974 -0.256314  0.006684  0.377421
3722 -0.492506 -0.188083  0.032107  0.114779 -0.293282 -0.010562  0.382953
3723 -0.513464 -0.166978 -0.028596 -0.079819 -0.233050  0.016055  0.370581
3724 -0.513334 -0.226689 -0.041153 -0.073539 -0.167929 -0.074490  0.374459
3725 -0.528842 -0.271615 -0.039601 -0.138861 -0.052670 -0.239873  0.373812

[3726 rows x 7 columns]

array([0.3955026 , 0.27142359, 0.14499078, 0.0681122 , 0.04809718,
       0.03312332])

Using 6/11 components 96.12% of the variance is retained
The most important features from the PCA analysis are
```

[30]: [(0, 'WindActual', -0.49639642720201316),
 (1, 'SystemDemand', 0.5981566993296759),
 (2, 'InterNet', 0.9561566550780313),
 (3, 'SysFrequency', -0.9493410379069294),
 (4, 'Fuel_Mixed', 0.4553419664220473),
 (5, 'ImbalanceSettlementPrice', 0.8786806428255461)]

[9]:
```
pca = PCA(n_components=0.9999)
pca_components = pca.fit_transform(pca_features)
pca_variances = pca.explained_variance_ratio_
plt.figure()
components = range(1, len(pca.explained_variance_ratio_) + 1)
plt.bar(components, pca.explained_variance_ratio_, align='center',␣
 ↪label='Individual Component Variance')
plt.plot(components, np.cumsum(pca.explained_variance_ratio_),␣
 ↪label='Cumulative Explained Variance', color='red')
plt.title('PCA Analysis on Input Features')
plt.ylabel('Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



1.3.1 Training Data

Given the input features have been identified we can export our training data for later use.

Feature Selection We're going to use 5 input features to start with 1. ImbalanceSettlmentPrice p_t 2. InterNet k_t 3. SysFrequency q_t 4. SystemDemand d_t 5. WindActual ω_t

Based on the combined Gray's analysis and dimensionality reduction using PCA. Interestingly enough, it seems that the time of the day (`day_cos day_sin` information) is not relevant to the prediction. Probabably because the underlying system information contributes more to the MEF than the time of day.

[10]:
```
scaler = MinMaxScaler()
training_data = df.copy()
training_data = training_data.reindex(columns=most_important_cols +␣
 ↪['MarginalEmissions'])
training_data[most_important_cols] = scaler.
 ↪fit_transform(training_data[most_important_cols].values)
training_data.to_csv(k.PROCESSED_DATA_DIR / 'features_target.csv')
training_data
```

[10]:

	WindActual	SystemDemand	InterNet	SysFrequency	Fuel_Mixed
0	0.520729	0.255325	0.628191	0.52	0.120529
1	0.512915	0.208629	0.482979	0.36	0.116270
2	0.499674	0.165483	0.577128	0.44	0.113239
3	0.456479	0.140360	0.482979	0.56	0.111701
4	0.445843	0.116876	0.482979	0.60	0.113225
…	…	…	…	…	…
3730	0.638811	0.414528	0.482979	0.56	0.043598
3731	0.614934	0.373293	0.562766	0.32	0.045528
3732	0.641849	0.337520	0.484574	0.52	0.042322
3733	0.608205	0.297925	0.482979	0.52	0.043318
3734	0.587367	0.270071	0.482979	0.60	0.043018

	ImbalanceSettlementPrice	MarginalEmissions
0	0.083333	311.846154
1	0.036406	1139.666667
2	0.047973	297.983871
3	0.036338	66.664122
4	0.038633	211.000000
…	…	…
3730	0.382420	111.531915
3731	0.368217	680.268293
3732	0.379670	-591.666667
3733	0.266953	-193.000000
3734	0.069653	-259.500000

```
[3726 rows x 7 columns]
```

1.4 Baseline Model

This is obviously going to be terrible - but it's the simplest model we can create.

The goal is to design a *linear model* that takes the carbon intensity and generation at time step t to predict the marginal emissions factor at the next. This is a MLP ANN without a hidden layer

$$p_t w_3 + k_t w_4 + q_t w_5 + d_t w_6 + \omega_t w_7 = ME_{t+1}$$

[11]:
```
# TODO develop baseline model
```