TA後形状の予測

作成日：　2024.12.9

データ元：　工程内形状データ、ライン作業実績

解析手順：　データ抽出・前処理　⇒機械学習（ランダムフォレスト）⇒予測　⇒グラフ化

1. データ抽出

In [1]:
```
import sqlite3
import pandas as pd

# ------------------------------------------------------------
# 抽出対象
# ------------------------------------------------------------
user_code = "D250"
thickness = 30.0
width = 495


# ------------------------------------------------------------
# データベース接続＆抽出
# ------------------------------------------------------------
dbname = "F:\\金属箔事業部\\秘L\\旧箔\\箔\\品質保証Gr\\QC150_品質データ\\010_オフライン形状測定装置データ_工場\\新形状測定装置\\★SQLi
#dbname = "./DB/keijyo_new.db"

# CASE WHEN THEN: 文字列の置換
# MAX : GROUP BY条件のなかで、m_timeの最大値を抽出

with sqlite3.connect(dbname) as conn1:
    sql = f'''
            SELECT
                * ,
                MAX(m_time),
                CASE
                    WHEN position = 'f' THEN 'F'
                    WHEN position = 'Ｆ' THEN 'F'
                    WHEN position = 'b' THEN 'B'
                    WHEN position = 'Ｂ' THEN 'B'
                    ELSE position
                END AS new_position
            FROM tbl_01
            WHERE
                (customer = "{user_code}")
                AND (thickness = "{thickness}")
                AND (width = "{width}")
                AND (treat_code = "4" OR treat_code = "9")
            GROUP BY
                coil_num,
                coildiv_num,
                ope_code,
                treat_code,
                CASE
                    WHEN position = 'f' THEN 'F'
                    WHEN position = 'Ｆ' THEN 'F'
                    WHEN position = 'b' THEN 'B'
                    WHEN position = 'Ｂ' THEN 'B'
                    ELSE position
                END
            ORDER BY m_date ASC
        '''
    df = pd.read_sql(sql, con=conn1)


# ------------------------------------------------------------
# DF作成
# ------------------------------------------------------------

# コイル番号
df["new_coil"] = df["coil_num"] + "-"+ df["coildiv_num"]

# 処理コードと位置の新規列　["9F","9B","4F","4B"] = ["FCR_F","FCR_B","TA_F","TA_B"]
df["process_position"] = df["treat_code"] + df["new_position"]

# 使用する列の指定
new_col = ["m_date", "new_coil", "process_position"] + [f"i{i}" for i in range(1, 21)]
```

```
df = df[new_col]

# CSV出力
#df.to_csv("data.csv", encoding='shift-jis')

df.head(3)
```

Out[1]:

	m_date	new_coil	process_position	i1	i2	i3	i4	i5	i6	i7	...	i11	i12	i13
0	2023-04-15 00:00:00	49134-11	9B	18.136928	3.790268	0.0	0.558058	0.992577	2.033761	2.587749	...	0.333904	0.013008	0.307649
1	2023-04-15 00:00:00	49134-11	9F	14.397729	3.015333	0.0	0.479469	0.775821	1.638432	2.051087	...	0.772083	0.248304	0.395933
2	2023-04-15 00:00:00	49134-12	9B	12.868052	4.347836	0.0	0.476016	1.263866	1.913309	2.823925	...	0.959822	0.789794	1.568435

3 rows × 23 columns

In [2]:
```
df_MF = df[df["process_position"] == "9F"]    # FCR F
df_MB = df[df["process_position"] == "9B"]    # FCR B
df_TF = df[df["process_position"] == "4F"]    # TA F
df_TB = df[df["process_position"] == "4B"]    # TA B

df_MF.columns = ["date_FCR_F", "new_coil", "pp_FCR_F"] + [f"FCR_F_{i}" for i in range(1, 21)]
df_MB.columns = ["date_FCR_B", "new_coil", "pp_FCR_B"] + [f"FCR_B_{i}" for i in range(1, 21)]
df_TF.columns = ["date_TA_F", "new_coil", "pp_TA_F"] + [f"TA_F_{i}" for i in range(1, 21)]
df_TB.columns = ["date_TA_B", "new_coil", "pp_TA_B"] + [f"TA_B_{i}" for i in range(1, 21)]

# "new_coil"をキーに各DFを結合
df_a1 = pd.merge(df_MF, df_MB, on="new_coil", how='left')
df_a1 = pd.merge(df_a1, df_TF, on="new_coil", how='left')
df_a1 = pd.merge(df_a1, df_TB, on="new_coil", how='left')

#df2.to_csv("data2.csv", encoding='shift-jis')
```

```
df_a1.head(3)
```

Out[2]:

	date_FCR_F	new_coil	pp_FCR_F	FCR_F_1	FCR_F_2	FCR_F_3	FCR_F_4	FCR_F_5	FCR_F_6	FCR_F_7	...	TA_B_11	TA_B_12	TA_B_13
0	2023-04-15 00:00:00	49134-11	9F	14.397729	3.015333	0.000000	0.479469	0.775821	1.638432	2.051087	...	2.006524	0.570906	0.501261
1	2023-04-15 00:00:00	49134-12	9F	15.557247	4.253771	0.108134	0.819745	1.503837	1.987190	2.420420	...	1.252476	0.168589	0.007054
2	2023-05-07 00:00:00	49135-1	9F	14.571781	5.226943	1.243542	1.725209	0.962206	0.626832	0.069162	...	2.305909	1.347314	0.940886

3 rows × 89 columns

熱処理条件

ライン作業実績のCSV読み込み

In [3]:
```
import cx_Oracle


# ------------------------------------------------------------------
# サーバー接続設定
# ------------------------------------------------------------------
SvrName = "WMDBMH21"
Port  = 1521
SvsName = "MSGK"
USRName = "HAKU1"
PWD = "HAKU1"

# ------------------------------------------------------------------------
# 日報表示　データ前処理関数
# ------------------------------------------------------------------------
def make_df(opt_date, customer, thickness):          # 引数： opt_date：作業年月日時分

    data_list = []
```

```
# 箔システム(MSGK)からデータ抽出
try:
    # OracleDBとの接続
    tns = cx_Oracle.makedsn(SvrName, Port, SvsName)
    conOcl = cx_Oracle.connect(USRName, PWD, tns)

    curOcl = conOcl.cursor()
    curOcl.arraysize = 1000

    # ライン作業実績TBLから期間が一致する項目を読み込み
    sql_Ocl = f'''
        SELECT
            HAKU.ライン作業実績_V.協定仕様番号_需要家コード,
            HAKU.ライン作業実績_V.協定仕様番号_品種コード,
            HAKU.ライン作業実績_V.協定仕様番号_鋼種コード,
            HAKU.ライン作業実績_V.協定仕様番号_調質コード,
            HAKU.ライン作業実績_V.協定仕様番号_仕上コード,
            HAKU.ライン作業実績_V.協定仕様番号_公称板厚,
            HAKU.ライン作業実績_V.協定仕様番号_公称板幅,
            HAKU.ライン作業実績_V.協定仕様番号_連番,
            HAKU.ライン作業実績_V.協定仕様番号_改訂番号,
            HAKU.ライン作業実績_V.コイル番号,
            HAKU.ライン作業実績_V.コイル分割番号,
            HAKU.ライン作業実績_V.作業年月日時分,
            HAKU.ライン作業実績_V.工程コード_作業コード,
            HAKU.ライン作業実績_V.工程コード_ラインコード1,
            HAKU.ライン作業実績_V.工程コード_ラインコード2,
            HAKU.ライン作業実績_V.工程コード_処理種別,
            HAKU.ライン作業実績_V.前面出し区分,
            HAKU.ライン作業実績_V.作業後コイル長さ,
            HAKU.ライン作業実績_工程個別_V.熱処理_速度,
            HAKU.ライン作業実績_工程個別_V.熱処理_張力入側,
            HAKU.ライン作業実績_工程個別_V.熱処理_張力中央,
            HAKU.ライン作業実績_工程個別_V.熱処理_張力出側,
            HAKU.ライン作業実績_工程個別_V.熱処理_H2流量,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温1,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温2,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温3,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温4,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温5,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温6,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温7,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温8,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温9,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温10,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温11,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温12,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温13,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温14,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温15,
            HAKU.ライン作業実績_工程個別_V.熱処理_炉温16
        FROM
            HAKU.ライン作業実績_V
            LEFT JOIN HAKU.ライン作業実績_工程個別_V
            ON (HAKU.ライン作業実績_V.コイル番号 = HAKU.ライン作業実績_工程個別_V.コイル番号)
            AND (HAKU.ライン作業実績_V.コイル分割番号 = HAKU.ライン作業実績_工程個別_V.コイル分割番号)
            AND (HAKU.ライン作業実績_V.工程順 = HAKU.ライン作業実績_工程個別_V.工程順)
            AND (HAKU.ライン作業実績_V.工程順子番 = HAKU.ライン作業実績_工程個別_V.工程順子番)
        WHERE
            (HAKU.ライン作業実績_V.作業年月日時分 >= '{opt_date}')
            AND (HAKU.ライン作業実績_V.前面出し区分 = '0')
            AND (HAKU.ライン作業実績_V.工程コード_作業コード = '2')
            AND (HAKU.ライン作業実績_V.工程コード_処理種別 = '4')
            AND (HAKU.ライン作業実績_V.協定仕様番号_需要家コード = '{customer}')
            AND (HAKU.ライン作業実績_V.協定仕様番号_公称板厚 = '{thickness}')
        ORDER BY
            ライン作業実績_V.作業年月日時分
    '''

    # カーソルを実行
    curOcl.execute(sql_Ocl)

    # リストとして全件取得し、データフレームに変換
    data_list = curOcl.fetchall()

# Oracleエラー
except cx_Oracle.OperationalError as e:
    error, = e.args
    print("Oracle関連のエラーが発生しました。")
    print(f"app4: Oracle ErrorCode: {error.code}")
    print(f"app4: Oracle ErrorMessage: {error.message}")
```

```
    # 一般エラー
    except Exception as e:
        # エラーメッセージの表示
        print(f"app4: Error Occurred: {e}")

    finally:
        #  DB接続の終了
        if conOcl is not None:
            curOcl.close()
            conOcl.close()


    df_ocl = pd.DataFrame(data_list)

    df_ocl.columns = ["需要家コード",
                      "品種コード",
                      "鋼種コード",
                      "調質コード",
                      "仕上コード",
                      "公称板厚",
                      "公称板幅",
                      "連番",
                      "改訂番号",
                      "コイル番号",
                      "コイル分割番号",
                      "作業年月日時分",
                      "作業コード",
                      "ラインコード1",
                      "ラインコード2",
                      "処理種別",
                      "前面出し区分",
                      "作業後コイル長さ",
                      "熱処理速度",
                      "張力入側",
                      "張力中央",
                      "張力出側",
                      "H2流量",
                ] + [f"炉温{i}" for i in range(1, 17)]

    return df_ocl
```

```
df_b1 = make_df("202304010000", "D250", "0300")

# コイル番号
df_b1["new_coil"] = df_b1["コイル番号"] + "-"+ df_b1["コイル分割番号"]

# 使用する列の指定
new_col = ["作業年月日時分", "new_coil", "熱処理速度"] + [f"炉温{i}" for i in range(1, 17)]
df_b1 = df_b1[new_col]

df_b1.sample(3)
```

Out[3]:

	作業年月日時分	new_coil	熱処理速度	炉温1	炉温2	炉温3	炉温4	炉温5	炉温6	炉温7	炉温8	炉温9	炉温10	炉温11	炉温12	炉温13	炉温14	炉温15	炉温16
125	202409142145	53555-1	40.0	657	654	656	654	656	655	649	656	512	403	298	301	208	194	98	29
117	202408210055	53302-22	52.5	656	655	657	654	655	655	650	656	516	403	305	301	206	194	106	31
100	202406220323	53166-2	52.5	658	655	656	654	654	655	651	656	515	419	301	301	212	189	94	26

In [4]:

```
# "new_coil"をキーに各DFを結合
df_c1 = pd.merge(df_a1, df_b1, on="new_coil", how='left')

df_c1.to_csv("data0.csv", encoding='shift-jis')

df_c1.sample(3)
```

Out[4]:

	date_FCR_F	new_coil	pp_FCR_F	FCR_F_1	FCR_F_2	FCR_F_3	FCR_F_4	FCR_F_5	FCR_F_6	FCR_F_7	...	炉温7	炉温8	炉温9	炉温10
112	2024-08-28 00:00:00	53474-1	9F	16.790717	3.791754	0.000000	0.358386	0.822472	2.051964	1.281495	...	649.0	656.0	511.0	403.0
99	2024-07-15 00:00:00	53298-2	9F	15.669783	5.561162	1.549341	0.803334	0.204147	0.522187	0.000000	...	650.0	656.0	516.0	414.0
125	2024-09-17 00:00:00	53577-22	9F	12.368234	2.039884	0.000000	0.354500	0.565953	1.347758	1.795442	...	NaN	NaN	NaN	NaN

3 rows × 107 columns

In [ ]:

2. 機械学習

「1.データ抽出」で出力された data.csvから、予測したいデータを除いて、data1.csvを作り、そのデータを読み込んで学習させる。

In [1]:
```
import pandas as pd

df = pd.read_csv("data1.csv", encoding='shift-jis')
df = df.dropna(subset=["TA_B_1"])
df.sample(3)
```

Out[1]:

	Unnamed: 0	date_FCR_F	new_coil	pp_FCR_F	FCR_F_1	FCR_F_2	FCR_F_3	FCR_F_4	FCR_F_5	FCR_F_6	...	炉温 7	炉温 8	炉温 9	炉温 10
153	153	2024/12/1 0:00	53904-2	9F	15.414498	2.517213	0.0	0.679142	0.889463	0.900861	...	650.0	655.0	516.0	406.0
0	0	2023/4/15 0:00	49134-11	9F	14.397729	3.015333	0.0	0.479469	0.775821	1.638432	...	643.0	654.0	506.0	406.0
69	69	2024/3/7 0:00	52518-22	9F	16.468021	4.562068	0.0	0.659055	1.154583	1.986549	...	651.0	656.0	513.0	403.0

3 rows × 108 columns

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error


# 特徴量とターゲットの設定
X = df[[f"FCR_F_{i}" for i in range(1, 21)] + ["熱処理速度"]]   # プロセス前のデータ
y = df[[f"TA_B_{i}" for i in range(1, 21)]]                    # プロセス後のデータ


# 学習関数
```

```
def learn(x, t, depth=3):
    # データの分割
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    model = RandomForestRegressor(n_estimators=100, random_state=0, max_depth = depth)
    model.fit(X_train, y_train)

    # モデル評価：MSE(Mean Squared Error)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)

    return round(mse, 3), model


# ハイパーパラメータ最適化（MSEの最小値の探索）
min_mse = float('inf')
best_depth = None

for j in range(1, 15):
    mse, model = learn(X, y, depth = j)
    print(f"depth:{j}, MSE:{mse}")

    if mse < min_mse:
        min_mse = mse
        best_depth = j

print(f"The depth with the lowest MSE is: {best_depth}")
```

```
depth:1, MSE:1.855
depth:2, MSE:1.804
depth:3, MSE:1.795
depth:4, MSE:1.817
depth:5, MSE:1.829
depth:6, MSE:1.818
depth:7, MSE:1.843
depth:8, MSE:1.873
depth:9, MSE:1.865
depth:10, MSE:1.825
depth:11, MSE:1.855
depth:12, MSE:1.851
depth:13, MSE:1.855
depth:14, MSE:1.853
The depth with the lowest MSE is: 3
```

In [3]:
```python
# データの分割
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# モデル適用（max_depthは上記の最小値を代入）
model = RandomForestRegressor(n_estimators=100, random_state=0, max_depth = best_depth)
model.fit(X_train, y_train)
```

Out[3]:

RandomForestRegressor

RandomForestRegressor(max_depth=3, random_state=0)

In [4]:
```python
# モデルの保存
import pickle

with open('RF01.pkl', 'wb') as f:
    pickle.dump(model, f)
```

In [ ]:

3. 予測

「1.データ抽出」で出力された data.csvから、予測したいデータだけを残してdata2.csvを作り、そのデータを使って予測する。

In [1]:
```
import pandas as pd

df = pd.read_csv("data2.csv", encoding='shift-jis')
df
```

Out[1]:

	Unnamed: 0	date_FCR_F	new_coil	pp_FCR_F	FCR_F_1	FCR_F_2	FCR_F_3	FCR_F_4	FCR_F_5	FCR_F_6	...	炉温7	炉温8	炉温9	炉温10	炉温11	
0	167	2024/12/10 0:00	53980-2	9F	15.807665	4.542202	0.122889	0.086451	0.004182	0.135352	...	648	656	513	403	303	3
1	168	2024/12/11 0:00	53981-21	9F	18.261842	3.529480	0.138316	0.247396	0.000000	0.146394	...	650	655	511	403	299	3
2	169	2024/12/11 0:00	53981-22	9F	15.921400	5.234024	0.366983	0.313617	0.000000	0.290176	...	649	656	514	404	304	3

3 rows × 108 columns

In [2]:
```
import pickle

with open('RF01.pkl', 'rb') as f:
    model = pickle.load(f)


# 予測用コイルのコイル番号取得
coil_research = input("予測するコイル番号を入力：")

# 予測用データのインデックスを取得
index_reserch = df[df["new_coil"] == coil_research].index[0]
```

```
# 予測用データをDF化
datalist = pd.concat([df.loc[index_reserch, "FCR_F_1":"FCR_F_20"], pd.Series(df.loc[index_reserch, "熱処理速度"], index=["熱処理
df_p = pd.DataFrame([datalist])

# 予測結果
predicted = model.predict(df_p)
df_TA_B_predicted = pd.DataFrame(predicted)
df_TA_B_predicted = df_TA_B_predicted.rename(index={0: 'TA_B_predicted'})
```

In [3]:
```
# FCR_F：入力データ
Li_FCR_F = df.loc[index_reserch, "FCR_F_1":"FCR_F_20"]
df_FCR_F = pd.DataFrame([Li_FCR_F])
df_FCR_F = df_FCR_F.reset_index(drop=True).rename({0: "FCR_F"})
df_FCR_F.columns = [ i for i in range(0, 20)]

# TA_B：実際のデータ
Li_TA_B_actual = df.loc[index_reserch, "TA_B_1":"TA_B_20"]
df_TA_B_actual = pd.DataFrame([Li_TA_B_actual])
df_TA_B_actual = df_TA_B_actual.reset_index(drop=True).rename({0: "TA_B_actual"})
df_TA_B_actual.columns = [ i for i in range(0, 20)]

# データフレームを結合
df2 = pd.concat([df_FCR_F, df_TA_B_actual, df_TA_B_predicted])
df2
```

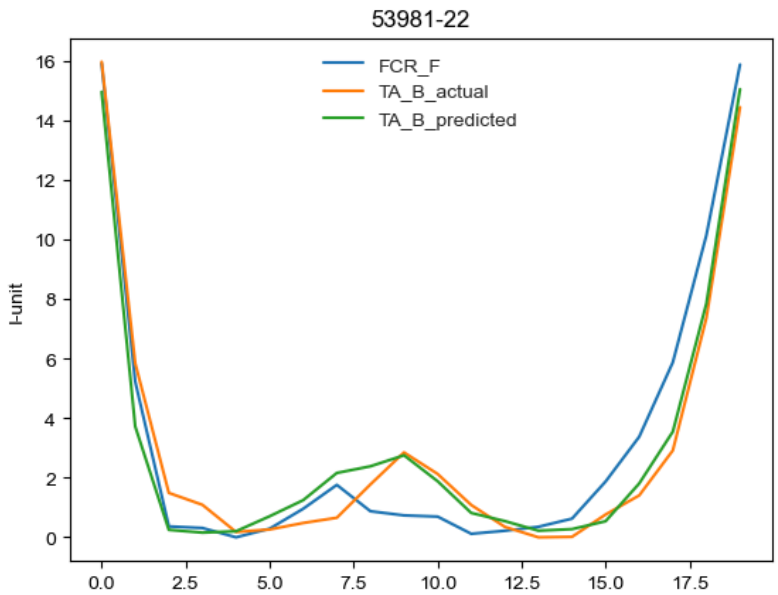
Out[3]:

	0	1	2	3	4	5	6	7	8	9	10	11	
FCR_F	15.921400	5.234024	0.366983	0.313617	0.000000	0.290176	0.960221	1.762699	0.877039	0.737137	0.696823	0.118812	0.
TA_B_actual	15.983786	5.850926	1.493249	1.092565	0.179748	0.263524	0.482008	0.657893	1.777459	2.854025	2.129814	1.086770	0.
TA_B_predicted	14.973967	3.725158	0.251152	0.157991	0.204010	0.707630	1.249488	2.163370	2.387228	2.757355	1.885082	0.819470	0.

グラフ描画

In [4]:
```
import matplotlib.pyplot as plt
import math
%matplotlib inline
```

```
fig, ax = plt.subplots()

# グリッドのスタイルを指定
plt.style.use('seaborn-v0_8-whitegrid')

# グラフ描画
for i in range(len(df2.index)):

    x  = df2.columns
    y  = df2.iloc[i]

    ax.plot(x, y, label = df2.index[i])
    ax.set_ylabel("I-unit")
    ax.set_title(coil_research)
    ax.legend(loc=0)

# グラフ保存・表示
plt.savefig(f"{coil_research}.png")
plt.show()
```



In [ ]: