Project Report: CO2 Injection Rate predictive analysis

Sapnil Sarker Bipro

Research Assistant, SPML Lab

Rajshahi University of Engineering & Technology, Rajshahi, Bangladesh.

Abstract: This study focuses on predicting the injection rate deltas (inj_diff) for carbon capture wells based on time series data and monitoring well parameters. To address this problem, a rigorous approach involving 15 distinct machine learning algorithms was utilized, alongside extensive feature engineering and feature selection processes. The selected models were carefully trained, evaluated, and compared to achieve optimal performance. An ensemble learning technique was employed to further enhance the prediction accuracy, achieving an impressive **R^2 score of 0.97** on the validation set. The model was successfully deployed on the iDareAI platform, providing a reliable tool for predicting injection rate deltas and validating carbon containment during the well injection process. The methodology developed serves as a crucial step towards monitoring and ensuring the integrity of carbon capture processes.

Problem Understanding: The problem involves predicting the deltas (inj_diff) in carbon capture well injection rates using time series data from well monitoring parameters. Accurate prediction of these deltas is crucial for validating carbon containment and detecting potential issues like CO_2 migration or leakage. By correlating injection rate changes with other well parameters, machine learning models are developed to forecast these variations. Effective feature engineering, selection, and model optimization are key to creating a robust solution for real-time monitoring of carbon capture processes. For this, participants were given 2 .csv files named:

1. CO2_Injection_rate train (for training)

2. CO2_Injection_rate test exam (for predicting)

The 2nd file had no annotation, so the first file's data was divided into training and validation subsets while solving the problem.

Feature Engineering: Initially, there were only 34 features. Using these features, **1114 features** were extracted manually. Broadly the feature types were: 1. temperature_diff_depths, 2. pressure_diff_depths, 3. lag, 4. absolute, and 5. trend features. The feature extraction code:

```python
def get_temperature_diff_depth(df):
    # Define depth levels and their corresponding column names
    depth_cols = {
        7061: 'Avg_VW1_Z01D7061Tp_F',
        6982: 'Avg_VW1_Z02D6982Tp_F',
        6837: 'Avg_VW1_Z04D6837Tp_F',
        6632: 'Avg_VW1_Z06D6632Tp_F',
        6416: 'Avg_VW1_Z07D6416Tp_F',
        5840: 'Avg_VW1_Z08D5840Tp_F',
        5482: 'Avg_VW1_Z0910D5482Tp_F',
        5653: 'Avg_VW1_Z09D5653Tp_F',
        5001: 'Avg_VW1_Z10D5001Tp_F',
        4917: 'Avg_VW1_Z11D4917Tp_F'
    }

    # Generate temperature differences dynamically
    for (depth1, depth2) in combinations(depth_cols.keys(), 2):
        col_name = f'Temperature diff {depth1}-{depth2} ft'
        df[col_name] = df[depth_cols[depth1]] - df[depth_cols[depth2]]

    return df
```

```python
def get_pressure_diff_depth(df):
    depth_columns = {
        7061: 'Avg_VW1_Z01D7061Ps_psi',
        6982: 'Avg_VW1_Z02D6982Ps_psi',
        6837: 'Avg_VW1_Z04D6837Ps_psi',
        6632: 'Avg_VW1_Z06D6632Ps_psi',
        6416: 'Avg_VW1_Z07D6416Ps_psi',
        5840: 'Avg_VW1_Z08D5840Ps_psi',
        5482: 'Avg_VW1_Z0910D5482Ps_psi',
        5653: 'Avg_VW1_Z09D5653Ps_psi',
        5001: 'Avg_VW1_Z10D5001Ps_psi',
        4917: 'Avg_VW1_Z11D4917Ps_psi',
    }

    # Compute pressure differences dynamically
    for (depth1, col1), (depth2, col2) in combinations(depth_columns.items(), 2):
        df[f'Pressure diff {depth1}-{depth2} ft'] = df[col1] - df[col2]

    return df
```

```python
def create_lag_features(df, features, n_lags=3):
    """
    Create lag features, percentage change, and difference features for the given dataframe and feature columns.
    """

    lagged_df = pd.DataFrame()

    # Ensure we only apply to numeric columns
    numeric_cols = df[features].select_dtypes(include=[np.number]).columns

    for feature in numeric_cols:
        for lag in range(1, n_lags + 1):
            # Create lagged features
            lagged_feature = f"{feature}_lag{lag}"
            lagged_df[lagged_feature] = df[feature].shift(lag)

            # Create percentage change features
            pct_change_feature = f"{feature}_pct_change{lag}"
            lagged_df[pct_change_feature] = df[feature].pct_change(lag)

            # Create difference features
            diff_feature = f"{feature}_diff{lag}"
            lagged_df[diff_feature] = df[feature].diff(lag)

    return lagged_df
```

```python
def create_abs_features(df, features):
    """
    Create absolute value and log absolute value features for the given dataframe and feature columns.
    """
    # Ensure we only apply to numeric columns (exclude datetime columns)
    numeric_cols = df[features].select_dtypes(include=[np.number]).columns

    for col in numeric_cols:
        df['Abs ' + col] = np.abs(df[col])
        df['Log abs ' + col] = np.log(np.abs(df[col]) + 1e-7)  # Adding small epsilon to avoid log(0)

    return df


def create_trend_features(df, features):
    """
    Create trend-based features such as Simple Moving Average (SMA) and Exponential Moving Average (EMA) for the given dataframe and
    """
    # Filter only numeric columns for rolling operations
    numeric_cols = df[features].select_dtypes(include=[np.number]).columns

    for col in numeric_cols:
        # Add simple moving average (SMA) with a window of 5
        df[col + ' sma 5'] = df[col].rolling(window=5).mean()

        # Add exponential moving average (EMA) with a span of 5
        df[col + ' ema 5'] = df[col].ewm(span=5, adjust=False).mean()

    return df
```

Using correlation matrix, 1092 features were excluded as the **rest 32 features** were correlated more than 80% with them. The selected features are:

['Avg_PLT_CO2VentRate_TPH', 'Avg_CCS1_WHCO2InjPs_psi', 'Avg_CCS1_ANPs_psi', 'Avg_CCS1_DH6325Ps_psi', 'Avg_CCS1_DH6325Tp_F', 'Avg_VW1_WBTbgPs_psi', 'Avg_VW1_WBTbgTp_F', 'Avg_VW1_ANPs_psi', 'Temperature diff 6982-6632 ft', 'Temperature diff 6837-5482 ft', 'Temperature diff 6837-5653 ft', 'Temperature diff 6837-4917 ft', 'Temperature diff 5482-5653 ft', 'Temperature diff 5482-4917 ft', 'Temperature diff 5653-4917 ft', 'Abs Rand', 'Log abs Avg_CCS1_WHCO2InjPs_psi', 'Abs inj_diff', 'Log abs inj_diff', 'Abs Diff pressure downhole-wellhead', 'Rand

sma 5', 'Avg_PLT_CO2VentRate_TPH sma 5', 'Avg_CCS1_ANPs_psi sma 5', 'Avg_VW1_WBTbgPs_psi sma 5', 'Avg_VW1_WBTbgTp_F sma 5', 'Avg_VW1_Z11D4917Ps_psi sma 5', 'inj_diff sma 5', 'inj_diff ema 5', 'Temperature diff 5482-4917 ft sma 5', 'Temperature diff 5653-4917 ft sma 5', 'Abs inj_diff sma 5', 'Abs Diff pressure downhole-wellhead sma 5']

Model Used: Using these selected features, 15 machine learning algorithms were used for the prediction of inj_diff. In this process a hold out cross validation was used, where 75% of the data were used for training and the rest are for validation. The results of these 15 models are depicted below:

Model	R^2 Score	MAE	RMSE
Decision Tree	0.962699	0.311896	0.863434
XGBoost	0.958825	0.352072	0.907168
Gradient Boosting	0.953519	0.453876	0.963849
CatBoost	0.906968	0.405766	1.363598
Random Forest	0.905890	0.456193	1.371474
Extra Trees	0.887268	0.463817	1.501041
LightGBM	0.851548	0.593398	1.722512
KNN	0.701302	0.929943	2.443351
Lasso Regression	0.686437	1.041088	2.503408
Bayesian Ridge	0.669998	1.175552	2.568192
Ridge Regression	0.669504	1.182068	2.570115
Linear Regression	0.669309	1.185001	2.570872
MLP Neural Network	0.660024	0.828186	2.606715
Elastic Net	0.443035	1.157364	3.336442
Support Vector Regression	0.188799	1.158614	4.026557

To improve the accuracy, a weighted ensemble model was deployed using the best 3 base models (decision tree (dt), XGBoost (xgb), and gradient boosting (gb)), and their weights were set as:

weight dt, weight xgb, weight gb = 0.34, 0.33, 0.33

The result of that ensemble model:

◆ Weighted Ensemble Model
R^2 Score: 0.9708
Mean Absolute Error: 0.3326
Root Mean Squared Error: 0.7644

Validation: Using the .pkl file of that ensemble model the prediction for the test set were done.

Conclusion: The weighted ensemble model achieved strong performance with an R^2 score of 0.9708, indicating high accuracy in predicting the injection rate deltas. The model's Mean Absolute Error (0.3326) and Root Mean Squared Error (0.7644) further highlight its robustness in minimizing prediction errors. This approach effectively combined multiple models, ensuring reliable predictions for carbon capture well monitoring.

References:

[1] https://github.com/luispintoc/ML-carbon-capture-challenge2023/blob/main/main.ipynb

[2] https://github.com/Vasanth1984/CO2MLChallenge_SPEGCS_2023/blob/main/02_MLCO2_LSTM_UnseedFinal.ipynb

[3] Wang, J., Qiang, X., Ren, Z., Wang, H., Wang, Y., & Wang, S. (2023). Time-Series Well Performance Prediction Based on Convolutional and Long Short-Term Memory Neural Network Model. *Energies*, *16*(1), 499.

[4] Iskandar, U. P., & Kurihara, M. (2022). Time-Series Forecasting of a CO_2-EOR and CO_2 Storage Project Using a Data-Driven Approach. *Energies*, *15*(13), 4768.

[5] https://www.spegcs.org/events/6655/

[6] https://github.com/luispintoc/ML-carbon-capture-challenge2023

IDAREAI SOLUTION: **https://idare.ai/whatif/sapnil-sarker-bipro/75e0320a-122c-4370-b8bd-8ffd78b160d7/?secret_key=V5HSRXhd88KXjCp270HSu7vrEaThng&fbclid=IwY2xjawIyyElleHRuA2FlbQIxMAABHXLMd5NghDsVUwb0m1sDClmXaz_b1gXuNhPywiZKTnwDaAG2NPz67H_TXw_aem_A5rlrhhCEmJ9J5mIXMxJVw**