03_baseline_model

March 10, 2024

1 Baseline Model: Random Forest Regressor

2 Import Modules

[1]:
```
import datetime
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.impute import SimpleImputer
```

3 Import Cleaned Datasets

[2]:
```
train_features = pd.read_csv('../data/clean/train_features.csv')
train_labels = pd.read_csv('../data/clean/train_labels.csv')
test_features = pd.read_csv('../data/clean/test_features.csv')
```

4 Identify Correlated Numeric Features

[3]:
```
# Define numeric features
numeric_features = [
    'year',
    'weekofyear',
    'ndvi_ne',
    'ndvi_nw',
    'ndvi_se',
    'ndvi_sw',
    'precipitation_amt_mm',
    'reanalysis_air_temp_k',
    'reanalysis_avg_temp_k',
    'reanalysis_dew_point_temp_k',
    'reanalysis_max_air_temp_k',
    'reanalysis_min_air_temp_k',
```

```
    'reanalysis_precip_amt_kg_per_m2',
    'reanalysis_relative_humidity_percent',
    'reanalysis_sat_precip_amt_mm',
    'reanalysis_specific_humidity_g_per_kg',
    'reanalysis_tdtr_k',
    'station_avg_temp_c',
    'station_diur_temp_rng_c',
    'station_max_temp_c',
    'station_min_temp_c',
    'station_precip_mm'
]
```

5 Preprocess and Split Data

[4]:
```
# Convert 'week_start_date' to datetime and extract 'year' and 'weekofyear'
train_features['week_start_date'] = pd.
 ↪to_datetime(train_features['week_start_date'])
train_features['year'] = train_features['week_start_date'].dt.year
train_features['weekofyear'] = train_features['week_start_date'].dt.
 ↪isocalendar().week

# Align train_features with train_labels
aligned_train_features = train_features.merge(
    train_labels,
    on=['city', 'year', 'weekofyear'],
    how='inner'
)

# Define features and labels
X = aligned_train_features[numeric_features + ['city']]  # Ensure that 'city'␣
 ↪is included for one-hot encoding
y = aligned_train_features['total_cases']

# Split the data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Preprocessing for numeric and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), ['city'])
    ]
)
```

6 Build Model Pipeline

For our first baseline model, we used Random Forest. This is an ensemble learning method based on decision trees. It operates by constructing a multitude of decision trees at training time and outputting the average prediction of the individual trees for regression tasks.

Random Forest makes few statistical assumptions about the data. It is a non-parametric method, meaning it does not assume a particular distribution for the data.

Also, Random Forest can capture complex nonlinear relationships in the data and can handle interactions between features without requiring explicit specification.

[5]:
```python
# Define the model
model = RandomForestRegressor(n_estimators=100, random_state=42)

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

7 Fit/Train Model

[6]:
```python
# Fit the model
pipeline.fit(X_train, y_train)
```

[6]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                  ['year', 'weekofyear',
                                                   'ndvi_ne', 'ndvi_nw',
                                                   'ndvi_se', 'ndvi_sw',
                                                   'precipitation_amt_mm',
                                                   'reanalysis_air_temp_k',
                                                   'reanalysis_avg_temp_k',
'reanalysis_dew_point_temp_k',
                                                   'reanalysis_max_air_temp_k',
                                                   'reanalysis_min_air_temp_k',
'reanalysis_precip_amt_kg_per_m2',
'reanalysis_relative_humidity_percent',
'reanalysis_sat_precip_amt_mm',
'reanalysis_specific_humidity_g_per_kg',
                                                   'reanalysis_tdtr_k',
                                                   'station_avg_temp_c',
                                                   'station_diur_temp_rng_c',
                                                   'station_max_temp_c',
                                                   'station_min_temp_c',
                                                   'station_precip_mm']),
                                                 ('cat', OneHotEncoder(),
                                                  ['city'])])),
                ('model', RandomForestRegressor(random_state=42))])
```

8 Validate Model

[7]:
```python
# Validate the model
val_predictions = pipeline.predict(X_val)
val_mae = mean_absolute_error(y_val, val_predictions)
print(f'Validation MAE: {val_mae}')
```

```
Validation MAE: 14.494212328767125
```

9 Preprocess Test Features

[8]:
```python
# Preprocess the test features in the same way as train features
test_features['week_start_date'] = pd.
 ↪to_datetime(test_features['week_start_date'])
test_features['year'] = test_features['week_start_date'].dt.year
test_features['weekofyear'] = test_features['week_start_date'].dt.isocalendar().
 ↪week

# Ensure the same features are used for the test set
test_X = test_features[numeric_features + ['city']]
```

10 Predict on Test Set

[9]:
```python
# Predict on the test set
test_predictions = pipeline.predict(test_X)
```

11 Make Submission

[10]:
```python
# Create submission DataFrame
submission = pd.DataFrame({
    'city': test_features['city'],
    'year': test_features['year'],
    'weekofyear': test_features['weekofyear'],
    'total_cases': test_predictions.astype(int)
})

# Date stamp for the filename
date_stamp = datetime.datetime.now().strftime("%Y-%m-%d")

# Save the submission
submission_filepath = f'../submissions/submission_{date_stamp}.csv'
submission.to_csv(submission_filepath, index=False)

print(f'Submission saved to {submission_filepath}')
print(f'Submission has {submission.shape[0]} rows')
```

```
Submission saved to ../submissions/submission_2024-03-10.csv
Submission has 416 rows
```

12 Submissions

This project uses Mean Absolute Error (`MAE`) as the evaluation metric. This is a measure of the average magnitude of the errors in a set of predictions, without considering their direction. It's calculated as the average of the absolute differences between the predicted values and the actual values.

12.1 First Submission

For our first submission, our model scored an `MAE` of `11.14`, which we calculated locally before submitting to DrivenData. This value indicates that, on average, our model's predictions on the validation set are off by approximately `11.14` cases. This was when we dropped all `NA` values from the dataset.

Our first submission was initially rejected by DrivenData because "Submission has `353` rows but should have `416`." We realised this is because we had dropped `NA` values from the training and test sets, but the test set should have kept the `NA` values. (In our third and final submission, we filled in `NA` values, which kept the number of samples in the train and test sets as they were originally.)

12.2 Second Submisison

For our second submission to DrivenData, we kept the `NA` values in training and test sets. We achieved a local validation `MAE` of `14.494212328767125` and DrivenData score of `25.6875`.

It's interesting that including `NA` values meant both our local `MAE` score is worse than the first submission, when we dropped `NA` values!

12.3 Next Steps

For our first two submissions, we used a **Random Forest** regressor. For our third and final submission, we used a **Negative Binomial** regression model, inspired by the benchmarking guide on DrivenData. To follow the next stage of the analysis, see the `04_baseline_model_revised.ipynb` notebook!