Tree Based Methods

April 30, 2020

[1]:
```
import numpy as np
import pandas as pd
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

import time
```

We first import the constructed files of predictors and response variables from R.

[2]:
```
group1_x = pd.read_csv("data/group1_lag3_x.csv")
group1_y = pd.read_csv("data/group1_lag3_y.csv")
```

[3]:
```
group2_x = pd.read_csv("data/group2_lag3_x.csv")
group2_y = pd.read_csv("data/group2_lag3_y.csv")
```

[4]:
```
group3_x = pd.read_csv("data/group3_cross_x.csv")
group3_y = pd.read_csv("data/group3_cross_y.csv")
```

[5]:
```
group4_x = pd.read_csv("data/group4_lag3_x.csv")
group4_y = pd.read_csv("data/group4_lag3_y.csv")
```

As both tree based methods are not affected by the scale of the variables, we do not need to standardize the data for either of the two methods shown below.

0.0.1 Gradient Boosted Trees

We use the sklearn function `GradientBoostingRegressor` along with `Multi Output Regressor` to adapt gradient boosting trees to a multi-target regression setting. We first need to tune the hyperparameters for the model: learning rate, number of iterations (known as `n_estimators` in the sklearn model), maximum depth of each tree, and whether we perform early stopping of the training or not. If we decide to allow for early stopping, we need to decide the number of iterations that will be used to decide. The default loss function is the squared loss.

Note that because of the lack of interpretability of Gradient Boosting Trees (and Random Forests later), it is very difficult to estimate why some combination of parameters performed better than

the others during the tuning process. However, what we can strive to achieve is check the relative importance of features for the final models that we arrive upon.

[45]:
```
def train_and_predict_boosting_tree(model, x, y, window_length):
    n_windows = x.shape[0] - window_length
    predictions = pd.DataFrame(0, index=range(n_windows), columns=y.columns[1:])
    y_actual = y.iloc[window_length:, 1:].reset_index()

    for i in range(n_windows):
        x_data = x.iloc[i:(i+window_length), 1:]
        y_data = y.iloc[i:(i+window_length), 1:]

        g_boost_multi = MultiOutputRegressor(model)
        g_boost_multi.fit(x_data, y_data)

        pred_x = x.iloc[(i+window_length):(i+window_length+1), 1:]
        pred_y = g_boost_multi.predict(pred_x)

        predictions.loc[i] = pred_y[0]

    sq_err = (y_actual.iloc[:, 1:]-predictions)**2
    feb_start = y.index[y["Date"] == '2020-02-01'][0]
    sq_err_feb_start = y_actual.index[y_actual["index"] == 152][0]

    mse_pre = sq_err.iloc[0:sq_err_feb_start, :].mean(axis=0)
    mse_post = sq_err.iloc[sq_err_feb_start:, :].mean(axis=0)
    mse_overall = sq_err.mean(axis=0)

    mse_df = pd.concat([mse_pre, mse_post, mse_overall], axis=1)

    return (predictions, mse_df)
```

We will first tune `learning_rate=0.1, 0.2, 0.5` and `n_estimators=50, 100, 200` in conjuction, as there is a trade-off between the two parameters.

[38]:
```
models_gbt = [
    GradientBoostingRegressor(loss='ls', learning_rate=0.01, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=100,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=200,␣
↪max_depth=3, n_iter_no_change=None),

    GradientBoostingRegressor(loss='ls', learning_rate=0.2, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.2, n_estimators=100,␣
↪max_depth=3, n_iter_no_change=None),
```

```
    GradientBoostingRegressor(loss='ls', learning_rate=0.2, n_estimators=200,␣
↪max_depth=3, n_iter_no_change=None),

    GradientBoostingRegressor(loss='ls', learning_rate=0.5, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.5, n_estimators=100,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.5, n_estimators=200,␣
↪max_depth=3, n_iter_no_change=None)
]
```

[43]:
```
mse_store = pd.DataFrame()
for model in models_gbt:
    predictions = train_and_predict_boosting_tree(model, group4_x, group4_y, 90)
    mse_store = pd.concat([mse_store, predictions], axis=1)
    print(predictions[1].iloc[:, 2].mean())
```

Next we try `max_depth=3, 5, 10` for the best model chosen above.

[ ]:
```
models_gbt = [
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=50,␣
↪max_depth=3, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=50,␣
↪max_depth=5, n_iter_no_change=None),
    GradientBoostingRegressor(loss='ls', learning_rate=0.1, n_estimators=50,␣
↪max_depth=10, n_iter_no_change=None)
]
```

[ ]:
```
mse_store = pd.DataFrame()
for model in models_gbt:
    predictions = train_and_predict_boosting_tree(model, group3_x, group3_y, 90)
    mse_store = pd.concat([mse_store, predictions], axis=1)
    print(predictions.iloc[:, 2].mean())
```

0.0.2 Random Forest

Now, we look at training a Random Forest over the given data. We have two hyperparamters to tune - maximum depth (`max_depth`) of each tree, and the number of trees (`n_estimators`). Note that the default criterion is to measure the quality of the split is the mean squared error, and the default setting is to take a bootstrap sample.

[66]:
```
def train_and_predict_random_forest(model, x, y, window_length):
    n_windows = x.shape[0] - window_length
    predictions = pd.DataFrame(0, index=range(n_windows), columns=y.columns[1:])
    y_actual = y.iloc[window_length:, 1:].reset_index()

    for i in range(n_windows):
```

```
        x_data = x.iloc[i:(i+window_length), 1:]
        y_data = y.iloc[i:(i+window_length), 1:]

        random_forest = model
        random_forest.fit(x_data, y_data)

        pred_x = x.iloc[(i+window_length):(i+window_length+1), 1:]
        pred_y = random_forest.predict(pred_x)

        predictions.loc[i] = pred_y[0]

    sq_err = (y_actual.iloc[:, 1:]-predictions)**2
    feb_start = y.index[y["Date"] == '2020-02-01'][0]
    sq_err_feb_start = y_actual.index[y_actual["index"] == feb_start][0]

    mse_pre = sq_err.iloc[0:sq_err_feb_start, :].mean(axis=0)
    mse_post = sq_err.iloc[sq_err_feb_start:, :].mean(axis=0)
    mse_overall = sq_err.mean(axis=0)

    mse_df = pd.concat([mse_pre, mse_post, mse_overall], axis=1)

    return predictions
```

We consider the following options for each parameter - `n_estimators=10, 100 (default), 200` and `max_depth=None (default), 3, 5, 10`.

[86]:
```
models_rf = [
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=None),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=10,␣
↪max_depth=10),

    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=None),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=50,␣
↪max_depth=10),

    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=None),
```

```
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=100,␣
↪max_depth=10),

    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=None),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=3),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=5),
    RandomForestRegressor(criterion='mse', bootstrap=True, n_estimators=200,␣
↪max_depth=10)
           ]
```

[91]:
```
mse_store = pd.DataFrame()
for model in models_rf:
    predictions = train_and_predict_random_forest(model, group4_x, group4_y, 90)
    mse_store = pd.concat([mse_store, predictions], axis=1)
    print(predictions.iloc[:, 2].mean())
```

0.0.3 Feature Importances

We calculate average feature importances to see what predictors are preferred. We look at pre- and post-Feb, as well as overall, to get an idea on how the structural break affects values. We plot barplots for all three cases as well.

Gradient Boosted Trees

[ ]:
```
final_models_gbt = [GradientBoostingRegressor(loss='ls', learning_rate=0.2,␣
↪n_estimators=100),
                    GradientBoostingRegressor(loss='ls', learning_rate=0.2,␣
↪n_estimators=50),
                    GradientBoostingRegressor(loss='ls', learning_rate=0.1,␣
↪n_estimators=100),
                    GradientBoostingRegressor(loss='ls', learning_rate=0.1,␣
↪n_estimators=50, max_depth=5)]
x_list = [group1_x, group2_x, group3_x, group4_x]
y_list = [group1_y, group2_y, group3_y, group4_y]
```

[ ]:
```
window_length = 90
k = 0

final_model_gbt_multi = MultiOutputRegressor(final_models_gbt[k])
x = x_list[k]
```

```
y = y_list[k]
n_windows = x.shape[0] - window_length
n_predictors = y.shape[1]-1
n_covariates = x.shape[1]-1

feature_imp_gbt = [[] for j in range(n_predictors)]
for i in range(n_windows):
    x_data = x.iloc[i:(i+window_length), 1:]
    y_data = y.iloc[i:(i+window_length), 1:]

    final_model_gbt_multi.fit(x_data, y_data)
    for j in range(n_predictors):
        feature_imp_gbt[j].append(final_model_gbt_multi.estimators_[j].
 ↪feature_importances_)
```

[ ]:
```
avg_feature_imp_gbt = np.mean(np.mean(feature_imp_gbt, axis=1), axis=0)
# Normalising them to sum up to 1
avg_feature_imp_gbt = avg_feature_imp_gbt/sum(avg_feature_imp_gbt)

avg_feature_imp_gbt_pre = np.mean([np.mean(i[0:62], axis=0) for i in␣
 ↪feature_imp_gbt], axis=0)
if sum(avg_feature_imp_gbt_pre) != 0:
    avg_feature_imp_gbt_pre = avg_feature_imp_gbt_pre/
 ↪sum(avg_feature_imp_gbt_pre)

avg_feature_imp_gbt_post = np.mean([np.mean(i[62:], axis=0) for i in␣
 ↪feature_imp_gbt], axis=0)
avg_feature_imp_gbt_post = avg_feature_imp_gbt_post/
 ↪sum(avg_feature_imp_gbt_post)
```

[ ]:
```
# sort labels in the same order as avg_feature_imp_gbt for graph labels
sorted_barplot_labels_gbt = [x for _, x in sorted(zip(avg_feature_imp_gbt,␣
 ↪range(n_covariates)))]
sorted_barplot_labels_gbt_pre = [x for _, x in␣
 ↪sorted(zip(avg_feature_imp_gbt_pre, range(n_covariates)))]
sorted_barplot_labels_gbt_post = [x for _, x in␣
 ↪sorted(zip(avg_feature_imp_gbt_post, range(n_covariates)))]
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_gbt)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_gbt,␣
 ↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Gradient Boosting Trees: Overall")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_gbt_pre)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_gbt_pre,␣
 ↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Gradient Boosting Trees: Pre Feb")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_gbt_post)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_gbt_post,␣
 ↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Gradient Boosting Trees: Post Feb")
plt.show()
```

Random Forests

[ ]:
```
final_models_rf = [RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=10, max_depth=10),
                   RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=200, max_depth=5),
                   RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=100, max_depth=3),
                   RandomForestRegressor(criterion='mse', bootstrap=True,␣
 ↪n_estimators=100, max_depth=5)]
x_list = [group1_x, group2_x, group3_x, group4_x]
y_list = [group1_y, group2_y, group3_y, group4_y]
```

[ ]:
```
window_length = 90
k = 2

final_model_rf = final_models_rf[k]
x = x_list[k]
y = y_list[k]
n_windows = x.shape[0] - window_length
n_predictors = y.shape[1]-1
n_covariates = x.shape[1]-1

feature_imp_rf = []
for i in range(n_windows):
    x_data = x.iloc[i:(i+window_length), 1:]
    y_data = y.iloc[i:(i+window_length), 1:]

    final_model_rf.fit(x_data, y_data)
    feature_imp_rf.append(final_model_rf.feature_importances_)
```

[ ]:
```
avg_feature_imp_rf = np.mean(feature_imp_rf, axis=0)
avg_feature_imp_rf = avg_feature_imp_rf/sum(avg_feature_imp_rf)
```

```
avg_feature_imp_rf_pre = np.mean(feature_imp_rf[0:62], axis=0)
if sum(avg_feature_imp_rf_pre) != 0:
    avg_feature_imp_rf_pre = avg_feature_imp_rf_pre/sum(avg_feature_imp_rf_pre)

avg_feature_imp_rf_post = np.mean(feature_imp_rf[62:], axis=0)
avg_feature_imp_rf_post = avg_feature_imp_rf_post/sum(avg_feature_imp_rf_post)
```

[ ]:
```
sorted_barplot_labels_rf = [x for _, x in sorted(zip(avg_feature_imp_rf,␣
↪range(n_covariates)))]
sorted_barplot_labels_rf_pre = [x for _, x in␣
↪sorted(zip(avg_feature_imp_rf_pre, range(n_covariates)))]
sorted_barplot_labels_rf_post = [x for _, x in␣
↪sorted(zip(avg_feature_imp_rf_post, range(n_covariates)))]
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_rf)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_rf,␣
↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Random Forest: Overall")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_rf_pre)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_rf_pre,␣
↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Random Forest: Pre Feb")
plt.show()
```

[ ]:
```
plt.bar(np.arange(n_covariates), list(sorted(avg_feature_imp_rf_post)))
plt.xticks(np.arange(n_covariates), sorted_barplot_labels_rf_post,␣
↪rotation='vertical')
plt.ylabel("Normalized Feature Importance")
plt.title("Random Forest: Post Feb")
plt.show()
```