Machine Learning Modelling- ARIMA

July 12, 2021

[ ]:
```
import pandas as pd
import numpy as np

pd.options.display.max_columns=100
```

[ ]:
```
df = pd.read_excel('apple_cost_forecasting.xlsx', header=1)
```

[ ]:
```
#rename cols

df.columns = ['part', 'program', 'alternate group', 'buy sell flag',␣
 ↪'commodity', 'm code', 'primary flag', 'allocation 1', 'buy price 1', 'sell␣
 ↪price 1', 'wac 1', 'allocation 2', 'buy price 2', 'sell price 2', 'wac 2',␣
 ↪'allocation 3', 'buy price 3', 'sell price 3', 'wac 3', 'allocation 4', 'buy␣
 ↪price 4', 'sell price 4', 'wac 4', 'allocation 5', 'buy price 5', 'sell␣
 ↪price 5', 'wac 5', 'allocation 6', 'buy price 6', 'sell price 6', 'wac 6',␣
 ↪'allocation 7', 'buy price 7', 'sell price 7', 'wac 7', 'allocation 8', 'buy␣
 ↪price 8', 'sell price 8', 'wac 8', 'allocation 9', 'buy price 9', 'sell␣
 ↪price 9', 'wac 9']
```

[ ]:
```
df.head()
```

1 Feature Selection and Transformation

[ ]:
```
wac_df = df_to_use.groupby(['part', 'program']).apply(pd.DataFrame)[['part',␣
 ↪'program', 'wac 1', 'wac 2', 'wac 3', 'wac 4', 'wac 5', 'wac 6', 'wac 7',␣
 ↪'wac 8', 'wac 9']]
wac_df.drop_duplicates(inplace=True)  #rows are duplicated when applying the␣
 ↪above fn

wac_df
```

1.1 Feature Transformation: Simulate WACs

[ ]:
```
def impute_with_mean(row):
    all_wacs = []
    for i in range(1, 10):
        all_wacs.append(row[f'wac {i}'])

    all_wacs_more_than_0 = [wac for wac in all_wacs if wac > 0]
    if all_wacs_more_than_0:  #'all_wacs_more_than_0' is NOT NA
        return pd.Series(all_wacs_more_than_0).mean()
    else:  #'all_wacs_more_than_0' is NA. occurs when the row has no wac that␣
 ↪is >0 (all its wacs are 0) so 'all_wacs_more_than_0' is an empty list
        return 0
```

[ ]:
```
#do the imputation

wac_df_imputed = pd.DataFrame(columns=['part', 'program', 'wac 1', 'wac 2',␣
 ↪'wac 3', 'wac 4', 'wac 5', 'wac 6', 'wac 7', 'wac 8', 'wac 9'])

for index in wac_df.index:
    row = wac_df.loc[index]  #gives a series
    mean_wac_for_row = impute_with_mean(row)

    for i in range(1, 10):
        if row[f'wac {i}'] == 0:
            row[f'wac {i}'] = mean_wac_for_row

    wac_df_imputed.loc[index] = row  #changes wac_df_imputed
```

[ ]:
```
wac_df_imputed
```

[ ]:
```
wac_df_imputed.columns=['part', 'program', 'wac_51', 'wac_52', 'wac_53',␣
 ↪'wac_54', 'wac_55', 'wac_56', 'wac_57', 'wac_58', 'wac_59']
```

[ ]:
```
std = wac_df_imputed.drop(['part', 'program', 'wac_59'], axis='columns').
 ↪std(axis=1)
mean = wac_df_imputed.drop(['part', 'program', 'wac_59'], axis='columns').
 ↪mean(axis=1)
```

[ ]:
```
import numpy as np
all_simulated_wacs = []

for i in range(len(std)):
    simulated_wacs_for_row = np.random.normal(mean.iloc[i], std.iloc[i], 50)
    all_simulated_wacs.append(simulated_wacs_for_row)
```

[ ]:
```
wac_df_imputed_simul = wac_df_imputed[['part', 'program']]

for i in range(50):
    col = []
    for simulated_wacs_for_row in all_simulated_wacs:
        col.append(simulated_wacs_for_row[i])

    wac_df_imputed_simul[f'sim_wac_{i+1}'] = col

wac_df_imputed_simul
```

[ ]:
```
wac_df.columns=['part', 'program', 'wac_51', 'wac_52', 'wac_53', 'wac_54',␣
↪'wac_55', 'wac_56', 'wac_57', 'wac_58', 'wac_59']
```

[ ]:
```
wac_df_for_model = pd.concat([wac_df_imputed_simul, wac_df[['wac_51', 'wac_52',␣
↪'wac_53', 'wac_54', 'wac_55', 'wac_56', 'wac_57', 'wac_58', 'wac_59']]],␣
↪axis=1)
```

[ ]:
```
wac_df_for_model
```

2 Tune Hyperparameters of ARIMA Model

2.1 Use grid search to determine best ARIMA values of (p, d, q)

2.1.1 "Best" defined as:

1) Lowest out-of-sample RMSE 2) Tightest confidence interval for WAC

[ ]:
```
#evaluate ARIMA model for a given (p, d, q) on test data

from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import math
import warnings
warnings.filterwarnings("ignore")

def evaluate_model(row, arima_order):
    X = np.array(row[2:58], dtype='float')  #sim_wac_1 to wac_56: used to train␣
↪model
    y = row[58:]  #wac_57 to wac_59: to be predicted by model

    model = ARIMA(X, order=arima_order)
    model_fit = model.fit()
    y_pred_df = model_fit.get_forecast(steps=3).summary_frame(alpha=0.05)
        #'steps' = number of time periods ahead to make forecast for
        #'alpha' = significance level used to create prediction interval for␣
↪the forecast
```

```
    #calculate weighted out-of-sample error (rmse): give highest weight to next␣
↪month's prediction bc our use case cares most about predicting next month's␣
↪WAC
    y_pred = y_pred_df['mean']

    next_month_squared_error = (y_pred[0] - y[0])**2
    two_months_from_now_squared_error = (y_pred[1] - y[1])**2
    three_months_from_now_squared_error = (y_pred[2] - y[2])**2

    weighted_mse = 0.7*next_month_squared_error + 0.
↪2*two_months_from_now_squared_error + 0.1*three_months_from_now_squared_error
    weighted_rmse = math.sqrt(weighted_mse)

    #calculate next month's ci width
    y_pred_df.fillna('NA', inplace=True)

    next_month_row = y_pred_df.iloc[0]

    if next_month_row['mean_ci_upper'] == 'NA':  #next month's predicted wac␣
↪has no prediction interval
        next_month_ci_width = 'NA'
    else:
        next_month_ci_width = next_month_row['mean_ci_upper'] -␣
↪next_month_row['mean_ci_lower']

    return weighted_rmse, next_month_ci_width
```

[ ]:
```
def try_combinations(row, p_values=range(0, 5), d_values=range(1, 5),␣
↪q_values=range(0, 2)):
    best_results = []

    for p in p_values:
        for d in d_values:
            for q in q_values:
                arima_order = (p, d, q)

                try:
                    weighted_rmse, next_month_ci_width = evaluate_model(row,␣
↪arima_order)
                    best_results.append([arima_order, weighted_rmse,␣
↪next_month_ci_width])
                except:
                    continue
```

```
    best_results_df = pd.DataFrame(best_results, columns=['arima order',␣
↪'weighted rmse', 'next months ci width'])
    best_results_with_ci_df = best_results_df[best_results_df['next months ci␣
↪width'] != 'NA']

    #obtain top 5 models = gives the 5 lowest weighted rmse
    top_5_best_results_with_ci_df = best_results_with_ci_df.
↪sort_values(by='weighted rmse', ascending=True)[:5]

    return top_5_best_results_with_ci_df
```

[ ]:
```
def fit_arima(row, arima_order):
    X = np.array(row[2:58], dtype='float')  #wac 1 to wac 56: used to train␣
↪model
    y = row[58:]  #wac 57 to wac 59: to be predicted by model

    model = ARIMA(X, order=arima_order)
    model_fit = model.fit()
    y_pred_df = model_fit.get_forecast(steps=3).summary_frame(alpha=0.05)
    y_pred_df.index = ['wac_57', 'wac_58', 'wac_59']
    y_pred_df = y_pred_df[['mean', 'mean_ci_lower', 'mean_ci_upper', 'mean_se']]

    y_results = pd.concat([y, y_pred_df], axis=1)
    y_results.columns = ['actual', 'predicted', 'prediction_lower',␣
↪'prediction_upper', 'prediction_sd']

    return y_results
```

[ ]:
```
def fit_optimal_arima(row, top_5_best_results_with_ci_df):  #fits on training␣
↪set of sim_wac_1 to wac_56, and gives optimal arima results on wac_57 to␣
↪wac_59

    #how to choose optimal arima order: out of the top 5 best models (lowest␣
↪weighted rmse), the optimal arima order is the one that leads to the lowest␣
↪ci width for next month's ci
    optimal_arima_order = top_5_best_results_with_ci_df.sort_values(by='next␣
↪months ci width', ascending=True).iloc[0]['arima order']
    optimal_weighted_rmse = top_5_best_results_with_ci_df.sort_values(by='next␣
↪months ci width', ascending=True).iloc[0]['weighted rmse']

    optimal_y_results = fit_arima(row, optimal_arima_order)

    return optimal_y_results, optimal_arima_order, optimal_weighted_rmse
```

[ ]:
```
#re-fit on all wacs up to wac 58, and test on wac 59
```

```
def test_optimal_arima(row, top_5_best_results_with_ci_df):
    X = np.array(row[2:60], dtype='float')  #sim wac 1 to wac 58: used to train␣
↪model
    y = row[60:].reset_index(drop=True)  #wac 59: to be predicted by model

    #optimal arima order is the one that gives the smallest ci width
    sorted_df = top_5_best_results_with_ci_df.sort_values(by='next months ci␣
↪width', ascending=True)

    for i in range(len(sorted_df)):
        optimal_arima_order = sorted_df.iloc[i]['arima order']

        try:
            model = ARIMA(X, order=optimal_arima_order)
            model_fit = model.fit()
            y_pred_df = model_fit.get_forecast(steps=1).summary_frame(alpha=0.
↪05)
            y_pred_df = y_pred_df[['mean', 'mean_ci_lower', 'mean_ci_upper',␣
↪'mean_se']]
            y_pred_df.fillna('NA', inplace=True)

            if y_pred_df['mean_ci_lower'] != 'NA':
                break
            else:
                continue

        except:
            continue

    y_results = pd.concat([y, y_pred_df, pd.Series(f'{optimal_arima_order}')],␣
↪axis=1)
    y_results.columns = ['actual', 'predicted', 'prediction_lower',␣
↪'prediction_upper', 'prediction_sd', 'optimal_arima_order']

    return y_results
```

2.2 Usage EG 1: Row 452 (WAC does not vary)

[ ]:
```
wac_df_for_model.iloc[452:453]
```

[ ]:
```
#conduct grid search for optimal arima order

best_results_df = try_combinations(wac_df_for_model.iloc[452])

best_results_df  #gives all arima orders that produce out-of-sample weighted␣
↪rmse < 0.005, which is our threshold for a good model
```

```
[ ]: optimal_y_results, optimal_arima_order, optimal_weighted_rmse =␣
     ↪fit_optimal_arima(wac_df_for_model.iloc[452], best_results_df)

     print(f'Optimal ARIMA order: {optimal_arima_order}')  #ordering: (AR value, I␣
     ↪value, MA value)
     print(f'Weighted RMSE: {optimal_weighted_rmse}')
```

```
[ ]: optimal_y_results
```

2.3 Usage EG 2: Row 0 (WAC varies a little)

```
[ ]: wac_df_for_model.iloc[0:1]
```

```
[ ]: #conduct grid search for optimal arima order

     best_results_df = try_combinations(wac_df_for_model.iloc[0])

     best_results_df  #gives all arima orders that have rmse < 0.005
```

```
[ ]: optimal_y_results, optimal_arima_order, optimal_weighted_rmse =␣
     ↪fit_optimal_arima(wac_df_for_model.iloc[0], best_results_df)

     print(f'Optimal ARIMA order: {optimal_arima_order}')  #ordering: (AR value, I␣
     ↪value, MA value)
     print(f'Weighted RMSE: {optimal_weighted_rmse}')
```

```
[ ]: optimal_y_results
```

```
[ ]: # use optimal arima model to re-fit on all wacs up to wac 58, and test on wac 59

     test_optimal_arima(wac_df_for_model.iloc[0], optimal_arima_order=(4, 1, 1))

     #=> model works: wac 59 will NOT be flagged out
```

2.4 Usage EG 3: Row 612 (WAC varies a lot)

```
[ ]: wac_df_for_model.iloc[612:613]
```

```
[ ]: #conduct grid search for optimal arima order

     best_results_df = try_combinations(wac_df_for_model.iloc[612])

     best_results_df  #gives all arima orders that have rmse < 0.005
```

```
[ ]: optimal_y_results, optimal_arima_order, optimal_weighted_rmse =␣
     ↪fit_optimal_arima(wac_df_for_model.iloc[612], best_results_df)

     print(f'Optimal ARIMA order: {optimal_arima_order}')  #ordering: (AR value, I␣
     ↪value, MA value)
     print(f'Weighted RMSE: {optimal_weighted_rmse}')
```

```
[ ]: optimal_y_results
```

```
[ ]: # use optimal arima model to re-fit on all wacs up to wac 58, and test on wac 59

     test_optimal_arima(wac_df_for_model.iloc[612], optimal_arima_order=(2, 1, 0))

     #=> model works: wac 59 will NOT be flagged out
```

3 Fit Model: 1 Model Per Component

```
[ ]: def optimal_arima_results_on_wac_59(row):
         top_5_best_results_with_ci_df = try_combinations(row)
         y_results = test_optimal_arima(row, top_5_best_results_with_ci_df)

         y_results.index=[f"({row['part']}, {row['program']})"]

         return y_results
```

```
[ ]: import time
     start = time.time()

     all_y_results = pd.DataFrame()

     for row_num in range(len(wac_df_for_model)):
         row = wac_df_for_model.iloc[row_num]

         row_y_results = optimal_arima_results_on_wac_59(row)

         all_y_results = pd.concat([all_y_results, row_y_results], axis=0)

         print(f'row {row_num} done out of 768')

     end = time.time()
     print(f'time elapsed: {(end-start)/60}')
```

```
[ ]: all_y_results
```

```
[ ]: all_y_results.to_csv('analysis per row.csv')
```

4 Model Performance Across All Components (Using MAPE)

[ ]:
```
def calc_absolute_perc_error(row):
    actual = row['actual']
    pred = row['predicted']

    if actual==0:
        return 0

    else:
        return np.abs((actual-pred)/actual)
```

[ ]:
```
all_y_results['absolute_perc_error'] = all_y_results.apply(lambda row:␣
↪calc_absolute_perc_error(row), axis=1)
```

[ ]:
```
all_y_results['absolute_perc_error'].mean()
```

[ ]:
```
all_y_results['absolute_perc_error'].std()
```

[ ]:
```
all_y_results['absolute_perc_error'].min()
```

[ ]:
```
all_y_results['absolute_perc_error'].max()
```

[ ]:
```
#find out how many (part, programs) have absolute perc error of 0

len(all_y_results[all_y_results['absolute_perc_error'] == 0])
```