1. Part 1

Firstly, I created the train set and the test set. As the value of the past instances of weather and cases is set to 30, only the data after day 31 will be selected (i.e. from day 31 to day 162 for the test set).

For the train set, `x_train` consists of `'max_temp'`, `'max_dew'`, `'max_humid'` and `'past_cases'` of past 10 days, and `y_train` is a list of the `daily_cases` of the corresponding day. The shape of `x_train` is (162, 40) and the shape of `y_train` is (162,).

Then train the model by `x_train` and `y_train`. In part1, we do not need to change the hyperparameters for the model.

The next step is processing the test data. The features in `x_test` but not in `x_train` will be dropped. Eventually, the features of `x_test` should be the same as those of `x_train`, and there are 40 features in total.

Finally, the model predicts the daily cases based on the features of `x_test` and return it.

2. Part 2

In part 2, it is allowed to design new approach and propose, aiming to boost the model performance and decrease the MAE of the test set.

Data pre-processing and tuning the hyperparameters of the model are useful ways to improve the performance of the model, which were adopted in this

specific task.

2.1 Data Pre-processing

2.1.1 Feature Selection

Since the size of the train data is relatively small, overfitting is a problem worthy attention. To avoid overfitting, as well as accelerate the training of the model, feature selection was adopted. I used `xgboost` to show the importance of the features, and it chould be found that the importance of the feature `'daily_cases'` is significantly greater than others(Figure 1). Thus it is reasonable to choose `'daily_cases'` as the only considered feature (but in a specific time series).

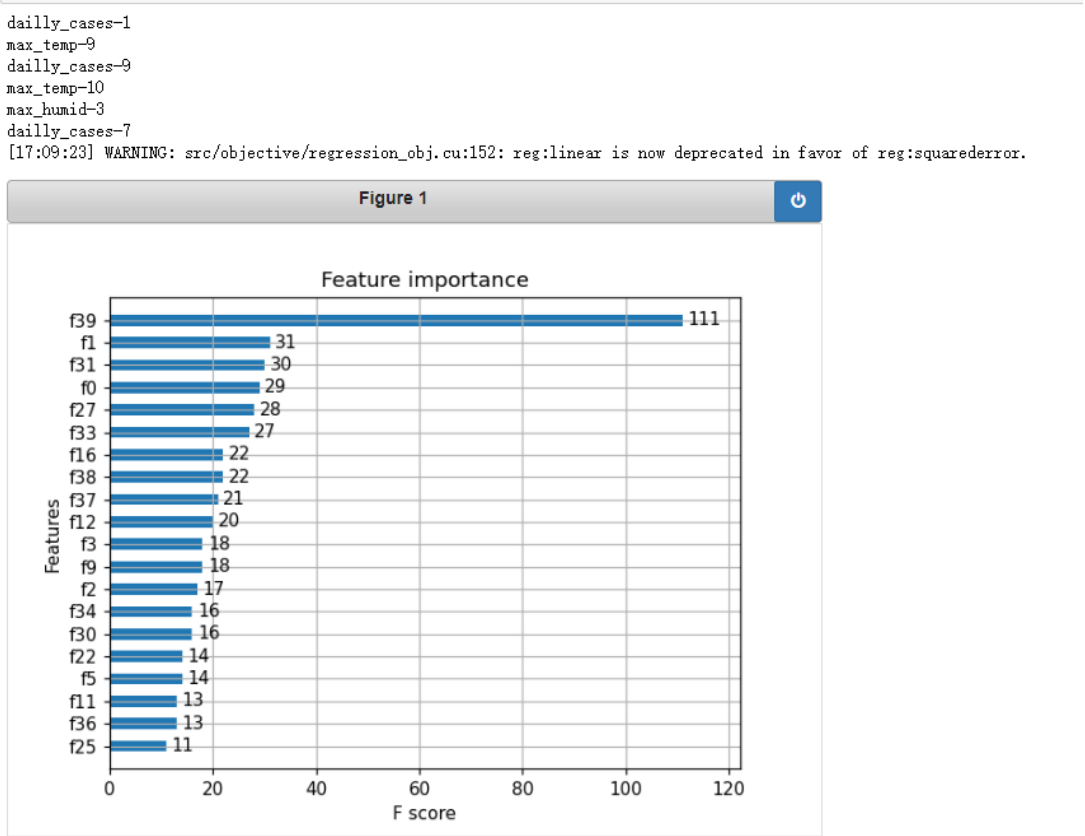

Figure 1 Feature Importance

2.1.2 Outlier

By observing the data, it could be found that the `daily_cases (y_train)`

for days before day 57 are much smaller than those in later days. These data could be regarded as outlier and could be dropped. In another words, only the data after day 57 would be considered.

2.1.3 past_cases_interval

Then I tried different values of `past_cases_interval` and got the best result when `past_cases_interval` was set to 19(Figure 2).

```
min_mae = 200
minimum_set = []
for beginday in np.arange(30, 33, 1):
    for pc_interval in np.arange(16, 20, 1):
            predicted_cases_part2 = []
            for idx in range(len(test_features)):
                test_feature = test_features.loc[idx]
                prediction = predict_COVID_part2(train_df, train_labels_df, test_feature, beginday,  pc_interval)
                predicted_cases_part2.append(prediction)
            #print(predicted_cases_part2)
            ## MeanAbsoluteError Computation...!

            MeanAbsError = mean_absolute_error(predicted_cases_part2, ground_truth)
            print(beginday,  pc_interval)
            print('MeanAbsError = ', MeanAbsError)
            if MeanAbsError < min_mae:
                min_mae = MeanAbsError
                minimum_set = [beginday, pc_interval]

print(min_mae)
print(minimum_set)
```

```
30 16
MeanAbsError =  70.8
30 17
MeanAbsError =  70.1
30 18
MeanAbsError =  68.4
30 19
MeanAbsError =  69.25
31 16
MeanAbsError =  70.75
31 17
MeanAbsError =  70.1
31 18
MeanAbsError =  68.8
31 19
MeanAbsError =  68.5
32 16
MeanAbsError =  71.05
32 17
MeanAbsError =  70.45
32 18
MeanAbsError =  69.65
32 19
MeanAbsError =  68.1
68.1
[32, 19]
```

Figure 2 past_cases_interval

2.1.4 Conclusion

As explained above, we only consider the data after day 57 (2.1.1), and the

`past_feature_interval` is set to 19 (2.1.3). Thus, the feature matrix should begin from day (57 + 19 = 76), and the features is the `daily_cases` of past 19 days.

```
begin_day = 76
past_cases_interval = 19
x_train = pd.DataFrame(columns = ['day'], data= [i for i in range(begin_day,len(train_df) + 1)])
y_train = train_labels_df.iloc[begin_day - 1:]
consider_features = ["dailly_cases"]

####processing train data

for feature in consider_features:
    for i in range(past_cases_interval,0, -1):
        n_col = feature+"-"+str(i)       #name of col, ***-10 to ***-1
        x_train[n_col] = -1              #init with value -1
        for idx in x_train.index:
            x_train.loc[idx,n_col] = train_df.iloc[idx + begin_day - 1 - i][feature]  #assign the value

train_features = x_train.columns.tolist()
test_features_list = test_feature.keys()
remove_list = []
for f in test_features_list:
    if f not in train_features:
        remove_list.append(f)
x_test = test_feature.drop(remove_list)

#######drop the col 'day'
x_train = x_train.drop(["day"], axis=1)
y_train = y_train["dailly_cases"]

#####convert to np.array
x_train = np.array(x_train)
y_train = np.array(y_train)
```

Figure 3 Data Pre-processing

2.2 Tuning Hyperparameters

After processing the data, I used **`GridSearchCV`** to tuning the hyperparameters for the SVM model (`scoring function : MAE, CV: 5-Folds CV`), then printed the best estimator (Figure 4, Figure 5).

```
def predict_COVID_part2(train_df, train_labels_df, test_feature):
    begin_day = 76
    past_cases_interval = 19
    x_train = pd.DataFrame(columns = ['day'], data= [i for i in range(begin_day,len(train_df) + 1)])  #day 31 - 162
    y_train = train_labels_df.iloc[begin_day - 1:]                                                    #day 31 - 162
    consider_features = ["dailly_cases"]
    svm_model = SVR()

    param_dict = {'kernel': ['poly'], 'gamma' :['scale'], "C": np.arange(5000, 12000, 2000), "coef0": np.arange(0.1, 0.7, 0.2),
                  "tol": [0.001], 'epsilon': [10, 11, 12], 'degree':[1,2]}
    grid_search = GridSearchCV(svm_model, param_dict, cv=5, scoring = 'neg_mean_absolute_error')


#     svm_model.set_params(**{'kernel': 'poly', 'degree': 1, 'C': 5000,
#                             'gamma': 'scale', 'coef0': 0.1, 'tol': 0.001, 'epsilon': 10})

    ####processing train data

    for feature in consider_features:
        for i in range(past_cases_interval,0, -1):
            n_col = feature+"-"+str(i)      #name of col, ***-10 to ***-1
            x_train[n_col] = -1             #init with value -1
            for idx in x_train.index:
                x_train.loc[idx,n_col] = train_df.iloc[idx + begin_day - 1 - i][feature]  #assign the value

#     train_features = x_train.columns.tolist()
#     test_features_list = test_feature.keys()
#     remove_list = []
#     for f in test_features_list:
#         if f not in train_features:
#             remove_list.append(f)
#     x_test = test_feature.drop(remove_list)



#     #######drop the col 'day'
    x_train = x_train.drop(["day"], axis=1)
    y_train = y_train["dailly_cases"]

    grid_search.fit(x_train,y_train)
    print(grid_search.best_estimator_)
    print(grid_search.best_score_)
```

Figure 4 GridSearchCV()

```
predicted_cases_part2 = []
test_feature = []
predict_COVID_part2(train_df, train_labels_df, test_feature)
# for idx in range(len(test_features)):
#     test_feature = test_features.loc[idx]
#     prediction = submission.predict_COVID_part2(train_df, train_labels_df, test_feature)
#     predicted_cases_part2.append(prediction)
# # MeanAbsoluteError Computation...!

# MeanAbsError = mean_absolute_error(predicted_cases_part2, ground_truth)
# print('MeanAbsError = ', MeanAbsError)
```

```
SVR(C=7000, coef0=0.30000000000000004, degree=1, epsilon=10, kernel='poly')
-417.50156174166034
```

Figure 5 Best Estimator

2.3 RESULTS

Using the processed data (2.1) and SVM model (parameters set as in 2.2), after training the model, the prediction on the test set could be made then.

The prediction and MAE is shown in Figure 6.

MAE = 62.25, which is much smaller than that in part1 (95.1).

```
    svm_model = SVR()
    svm_model.set_params(**{'kernel': 'poly', 'degree': 1, 'C': 7000,
                            'gamma': 'scale', 'coef0': 0.3, 'tol': 0.001, 'epsilon': 10})

    x_test = x_test.drop(['day'])
    svm_model.fit(x_train, y_train)
    x_test = [np.array(x_test)]
    return (math.floor(svm_model.predict(x_test)))
```

```
predicted_cases_part2 = []
test_feature = []
# predict_COVID_part2(train_df, train_labels_df, test_feature)
for idx in range(len(test_features)):
    test_feature = test_features.loc[idx]
    prediction = predict_COVID_part2(train_df, train_labels_df, test_feature)
    predicted_cases_part2.append(prediction)
# MeanAbsoluteError Computation...!
print(predicted_cases_part2)
MeanAbsError = mean_absolute_error(predicted_cases_part2, ground_truth)
print('MeanAbsError = ', MeanAbsError)
```

```
[938, 844, 943, 940, 914, 989, 1025, 875, 844, 892, 901, 869, 956, 928, 872, 774, 884, 880, 871, 951]
MeanAbsError =  62.25
```

Figure 6 Results

2.4 Reflection

The decrease of MAE shows that the model in part2 is better than that in part1 for this specific task. In another words, the approach in part2 could improve the performance of the SVM model.

As explained in 2, feature-selection could decrease the number of features, making the model pay more attention on the important feature and avoiding the overfitting.

`GridSearchCV()` is a practical method to find the best hyperparameters for the model. Using the '`poly`' kernel could create more poly-features for every single sample. The main parameters for poly kernel are `degree, gamma` and `coef0`. By cross validation, it could return the best estimator. However, due to the consideration of time cost, the number of the hyperparameters to be tuned is limited. It is important to strike a balance between the model's performance and the time cost.