<Project Report- COMP9318>

z5302513 / Kihwan Baek / COMP9318 Project Report

1. Part1

In part1, we used weather and past cases as features, especially [max_temp, max_dew, max_humid, past_cases], to predict the probable cases of COVID-19. Before we predict the cases, we needed to prepare and follow several steps to do that.

<pre-processing steps>

1. Constructing a feature matrix for training data based on the past instances of weather and past cases(N=30).

- At this step, there are 192 daily cases in training data. So, the trained data include the information of 30 past cases. For example, the first row is the 'day31' data which include information of day1~day30' in training set. Therefore, the number of row will be total 192-30 = 162.

- And also, there are 17 different features which include 'max_temp', 'avg_temp' etc.

  So, for each feature(such as 'max_temp), there will be the data of 30 days. Hence,

  the the number of total columns will be 17 * 30 = 510.

Following the steps, we made 162 * 510 dimensional feature matrix.

Secondly, we use just some information from this matrix which are [max_temp, max_dew, max_humid, past_cases] to make the training data. Hence, following past_weather_interval and past_cases_interval, we use only the features to predict other test cases.

For example, if past_weather_interval = past_cases_interval = 15, we use 15*1(past_cases) + 15*3 (weather_cases) = 60 features which are [past_15 … past_1….weather_15 … weather_1].

Finally, we use the 162 * (past_interval + weather_interval*3) matrix as the train_data and use it to predict the test cases. For modelling the train_data, we used 'SVR' which is 'Support Vector Regression'. In 'SVR' model, it find the function which minimize the error outside the margin.

So, we find the function using some features as the support vectors and finally, we predict the future target values based on given information.

2. Part2

In part2, for making the performance of the model improved, we need to consider some methods to do that.

- pre-processing
    - standardization
    - dealing with outliers
- feature-selection
    - feature selection method
    - case and weather interval
- find best hyper-parameters

Firstly, in terms of data pre-processing, the raw data is very messy and not organized. So, it can distort statistical analyses and hinder finding the right pattern. So, we need to clean and organize the data. In this project, I used **'RobustScaler()'** and **'MinMaxScaler()'**. Using 'RobustScaler', which use median value instead of mean value and scale data through IQR, we can minimize the effect of outliers. So, I chose the scaling method to minmize the effect of outliers. Also, using 'MinMaxScaler' again, I scaled the data between 0 and 1 to normalize.

The important thing to note in this process is that the training data include past_cases. It is the most important feature to predict target value and the correlation between the feature and target is the highest one. So, I used scaling by keeping the pase_cases unchanged.

```
scaler1 = RobustScaler()
scaled_df = scaler1.fit_transform(train_df.iloc[:,:-1])
scaled_df = pd.DataFrame(data=scaled_df,columns=train_df.columns[:-1])
train_df = pd.concat([scaled_df,train_df.iloc[:,-1]],axis=1)

min_max_scaler = MinMaxScaler()
scaled_df = min_max_scaler.fit_transform(train_df.iloc[:,:-1])
scaled_df = pd.DataFrame(data=scaled_df,columns=train_df.columns[:-1])
train_df =  pd.concat([scaled_df,train_df.iloc[:,-1]],axis=1)
```

Secondly, we need to find the best features to predict future cases. So, I used 'SelectFromeModel' using 'RandomForestClassifier' for feature selection which evaluate feature importances and select the most relevant features. As a result, it chose ['day', 'max_temp', 'avg_temp', 'avg_dew', 'avg_humid', 'min_humid', 'avg_wind_speed', 'dailly_cases'] for the best features.

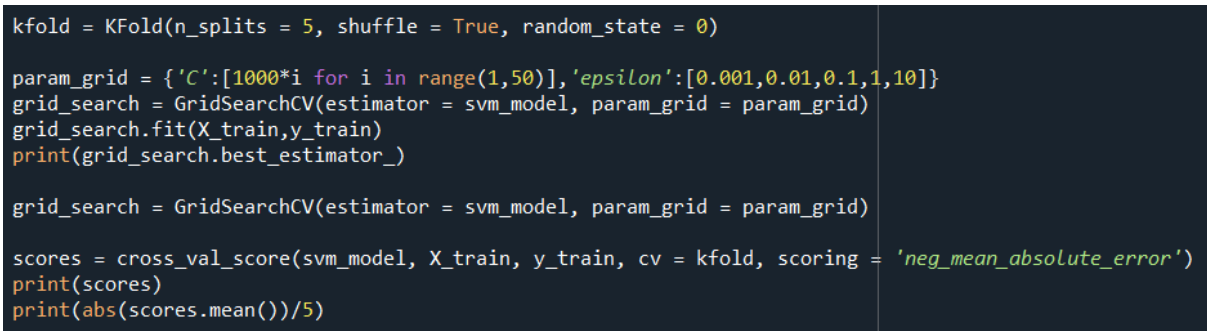
```
### Using SelectFromModel
sel_ = SelectFromModel(RandomForestClassifier(random_state = 1))
sel_.fit(train_df, train_labels_df)
feats_list = sel_.get_support()

RF_chosen = [i for i, x in enumerate(feats_list) if x]
chosen = train_df
chosen = pd.DataFrame(chosen)
chosen = chosen.columns[RF_chosen]
print(chosen)
```

```
Index(['day', 'max_temp', 'avg_temp', 'avg_dew', 'avg_humid', 'min_humid',
       'avg_wind_speed', 'dailly_cases'],
      dtype='object')
```

So, I chose the same features which are ['day', 'max_temp', 'avg_temp', 'avg_dew', 'avg_humid', 'min_humid', 'avg_wind_speed', 'dailly_cases'] as the result.

For the past cases interval and weather interval, I chose (past_cases = 10, past_weather = 9) that are almost the same as the previous one.

Finally, I used kfold method and grid search to test the model and find the best hyper-parameter for the target values.



```
kfold = KFold(n_splits = 5, shuffle = True, random_state = 0)

param_grid = {'C':[1000*i for i in range(1,50)],'epsilon':[0.001,0.01,0.1,1,10]}
grid_search = GridSearchCV(estimator = svm_model, param_grid = param_grid)
grid_search.fit(X_train,y_train)
print(grid_search.best_estimator_)

grid_search = GridSearchCV(estimator = svm_model, param_grid = param_grid)

scores = cross_val_score(svm_model, X_train, y_train, cv = kfold, scoring = 'neg_mean_absolute_error')
print(scores)
print(abs(scores.mean())/5)
```

These methods showed the best hyper-parameters and expected score depending on training data. So, I just chose best parameters using these methods ('C'=19000 and 'epsilon'=0.001).