Term Project Phase 2

Aya Mourad

November 18, 2020

Introduction

This report covers our progress in the term project. We have converted our time series to a regression problem by adding 5 lags as new features and augmenting season and trends as features. Consequently, we started experimenting our regression problem using different regression models including XGBoost, Neural Network (keras), Gradient Boosting Regressor, and Decision Tree Regressor.

Time Series Analysis

Time series data can be phrased as supervised learning. To do so multiple step have been taken into consideration.

Feature Engineering

We did a simple feature engineering, to get the date and month from the Date.

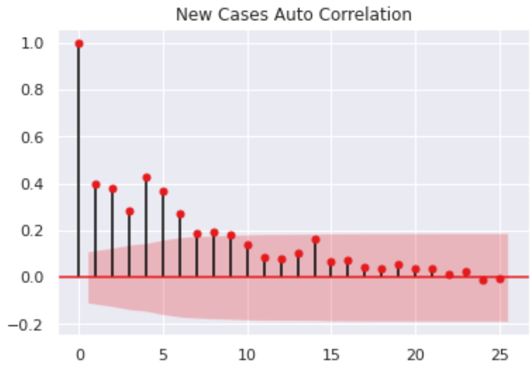
```
df['year'] = df.Date.apply(lambda x: x.year)
df['month'] = df.Date.apply(lambda x: x.month)
df['day'] = df['Date'].apply(lambda x: x.day)
```

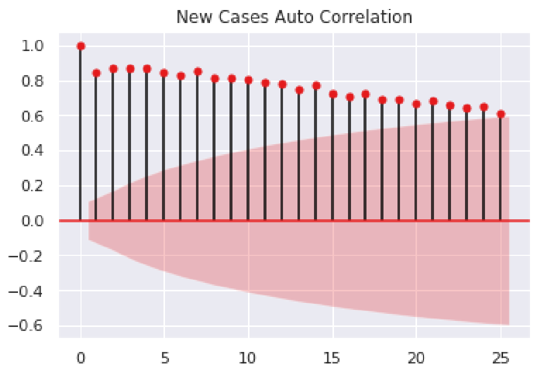
Lags and Auto Correlation Plots

Lag features are the classical way that time series forecasting problems are transformed into supervised learning problems.In order to determine our lags we first splitted our main datasets per country and studied the lags and time series components per each. We used autocorrelation plots to determine our number of lags. Most of the plots show a positive autocorrelation above the confidence intervals i.e. drawn as a cone. By default, this is set to a 95% confidence interval, suggesting that correlation values outside of this cone are very likely a correlation and not a statistical fluke. We show below some of the countries autocorrelation plots:

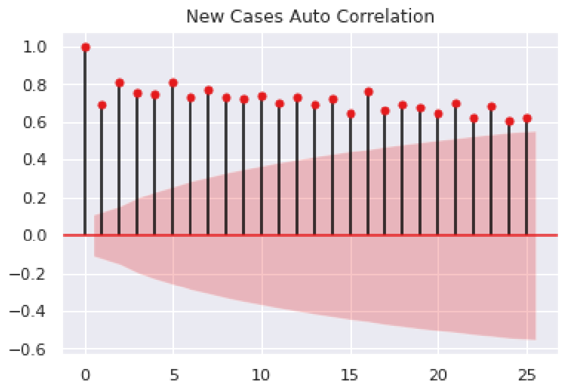
After analyzing the plots, we noticed that lag=5 is a good number to proceed in the conversion where the 5 lags were above the confidence interval for all the countries.

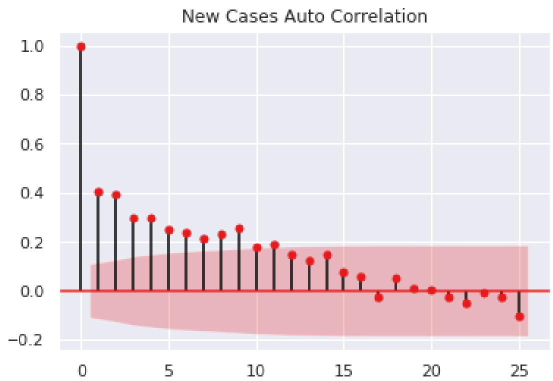
```
nb_lags=5
for country in (unique(df['Country'])):
  for i in range(1, nb_lags + 1):
    lag_label = 'lag_' + str(i)
    df_country=df_cases_country[country]
    df_country[lag_label] = df_country['new_cases'].shift(i)
    df_cases_country[country]=df_country
```





New Cases Auto Correlation





Time series Components

In this section, we used the statsmodel.api library to plot and retrieve the seasonal and trend components for each country, then we added both as new features.

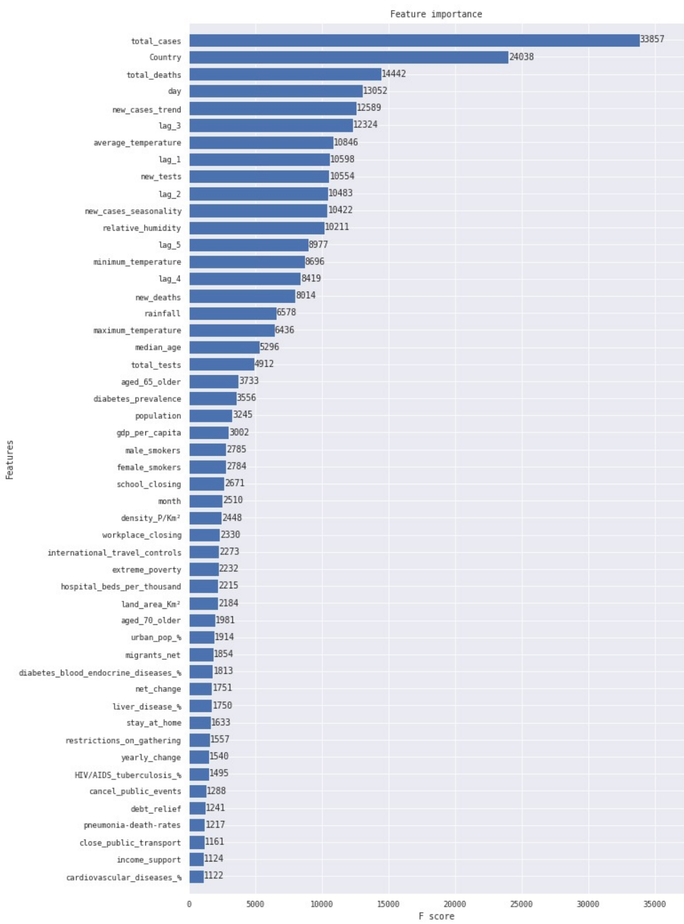
```
df_decomposition={}
def plot_seasonal_decomposition(df):
  fig_decomp = matplotlib.pyplot.figure(figsize=(8.0, 5.0))
  decomposed_cases_volume = sm.tsa.seasonal_decompose(df["new_cases"].
    values,model='additive',freq=7, extrapolate_trend='freq') # The
    frequency is set to 7 days
  plt=decomposed_cases_volume.plot()
  return decomposed_cases_volume
for country in (unique(df['Country'])):
  df_decomposition[country]=plot_seasonal_decomposition(df_cases_country
    [country])
for country in (unique(df['Country'])):
    trend = df_decomposition[country].trend
    season = df_decomposition[country].seasonal
    df_country=df_cases_country[country]
    target_variable='new_cases'
```

```
    df_country['%s_trend' % target_variable] = trend
    df_country['%s_seasonality' % target_variable] = season
    df_cases_country[country]=df_country
```

After converting our problem into a regression problem, we recombined the subsets(per country) as one dataframe referred to as **df_model** in the code.

Feature Selection

We repeated feature selection exercise using XGBoost with the new data including the lags, trends and seasonality. We noticed that the 5 lags, trend and seasonality are shown on the top 15 features.



Regression Models Metrics

Before delving into our models, we would like to define regression error metrics. First, we need to understand these metrics to evaluate our models and determine whether the results are accurate or misleading.

R2 Score

R2 score values are between 0 and 100%.

$$R^2 = 1 - \frac{\sum_i (y_{test} - y_{predicted})^2}{\sum_i (y_{test} - (\frac{\sum y_{test}}{n}))}$$

$$\mathrm{R}^2 = 1 - \frac{\text{Unexplained Variation}}{\text{Total Variation}}$$

R-Squared is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model, it is a statistical measure of how well the regression line approximates the actual data. A low percentage could indicate a non-valid regression model.

Mean Square Error

$$\mathrm{MSE} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} (\hat{\mathrm{y}}_{\mathrm{i}} - \mathrm{y}_{\mathrm{i}})^2$$

Mean square error is the average of the square of the error. The error is the difference between the predicted values y_predicted and the tested values y_test.The difference is squared so that negative and positive values do not cancel each other out.The lower the value the better and 0 means the model is perfect. We will use MSE's basic value in selecting one model over another. Root Mean Square Error is the square root of MSE, it can be used when MSE show a big number, where it would help us compare the models easily.

Mean Absolute Error

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| \,.$$

Mean absolute error is same as MSE, however instead of taking taking the square, it takes the absolute value of the error between y_predicted and y_test.

XGBoost Regressor Model

Fitting the model

For our first try for the model, we chose parameters at random and fit our model using KFold and cross_val_score.

```
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, learning_rate = 0.08,max_depth = 10, alpha = 10, n_estimators = 10)
kfold = KFold(n_splits=10, random_state=0)
results = cross_val_score(xg_reg, X_train, y_train, cv=kfold,scoring='r2')
xg_reg.fit(X_train,y_train)
y_pred = xg_reg.predict(X_test)
```

XGBoost Parameters

Relevant parameters

- booster: Select the type of model to run at each iteration
  - gbtree: tree-based models **(default parameter)**
  - gblinear: linear models
- nthread: default to maximum number of threads available if not set
- objective: This defines the loss function to be minimized

Parameters for controlling speed

- subsample: Denotes the fraction of observations to be randomly samples for each tree
- colsample_bytree: Subsample ratio of columns when constructing each tree.

- n_estimators: Number of trees to fit.

Important parameters which control overfiting

- learning_rate: Makes the model more robust by shrinking the weights on each step
- max_depth: The maximum depth of a tree.
- min_child_weight: Defines the minimum sum of weights of all observations required in a child.

Tuning Parameters

GridSearchCV params:

- estimator: estimator object
- param_grid : dict or list of dictionaries
- scoring: A single string or a callable to evaluate the predictions on the test set. For our problem we used r2 scoring to evaluate the model's performance and neg_mean_squared_error to study our learning curve. r2 assesses the goodness-of-fit of regression model on a scale from 0 to 1.
- n_jobs: Number of jobs to run in parallel.-1 means using all processors.
- cv: cross-validation, None, to use the default 3-fold cross validation. Integer, to specify the number of folds in a (Stratified) KFold. We used a KFold=10.

The below table shows all possible parameters we tuned for the XGBoost regressor model and the result of the best fit for each parameter.

XGBoost Hyper-Parameters Tuning		
Parameter	**Values**	**Best Parameters**
learning_rate	[0.5,0.55,0.6,0.7,0.75,0.8]	0.5
max_depth	[3, 5, 7, 10]	10
min_child_weight	[1, 3, 5]	5
subsample	[0.5, 0.7]	0.7
colsample_bytree	[0.5, 0.7]	0.7
n_estimators	[10,20, 30,40,50, 60]	50

Model fitting with best parameters

Based on the results from the hyper-parameter tuning, we fitted our model on the parameters and achieved the below evaluation metric.

Metric	Best Parameters Model
r2	0.97
RMSE	825.860722
MAE	141.9994091

Scaling

Decision trees and random forests models only needs to pick cut points on features to split a node. Splits are not sensitive to transformations where a split on one scale has a corresponding split on the transformed scale. We can notice that the scaling doesn't have a notable impact on the metrics. The small variation can be justified because of taking different X_train and X_test each time we were transforming and testing the transformation on our model metrics.

	XGBoost Scaling Metric				
	Standard Scaler	**MinMax Scaler**	**Robust Scaler**	**Normalized scaler**	**Quantile Transformer**
r2	0.98	0.97	0.97	0.97	0.97
RMSE	908.72	922.92	839.47	856.78	977.92
MAE	145.77	150.57	139.11	164.16	155.75

Learning Curve

In this section, we used learning_curve() library from sickit-learn to generate the learning curve of our XGBoost regression model. Many insights can deducted from the plot:

- **variance** can be defined as gap = validation error − training error. We can notice the wide gap between the validation learning curve and training learning curve which indicates high variance i.e. model fits training data too well causing problems in testing data. However, the gap is decreasing as the train size increases.

- **bias** Looking at the training curve, we can notice that as the training size increases, the validation error is slightly increasing. We have a low bias.

- **Over Fitting** The wide gap and the low validation error may indicate overfitting, where we can notice that the model performs well on the validation set and poorer in the training set (approx. 10^6 vs. approx. 10^3)
- **Continuous decrease** As the training size increases the validation error decreases, we can deduce that new instances are very likely to give better results. Also, the validation error curve tends to decrease, this indicates that it still can decrease more towards the training curve.

Training Size	Validation Scores	Training Scores
1934	1.716345	2.553669e+06
6288	109.570876	1.962481e+06
10641	298.060108	1.453077e+06
14995	540.652385	1.154362e+06
19349	806.582714	1.028261e+06

As a conclusion, our model suffers from high variance and low bias with overfitting, it can be improved by:

- Increase the training set which depends on the daily cases reported of COVID-19
- Increase the bias and decrease the variance by using less features.

XGBoost Feature Selection

Top features are retrieved using SelectFromModel from xgboost regressor with best parameters tuned. The threshold we took is 0.001. We repeated the steps done previously and below the summary of the results of each step.

Features Selected	'total_cases', 'total_deaths', 'new_deaths', 'new_tests', 'diabetes_prevalence', 'female_smokers', 'workplace_closing', 'cancel_public_events', 'restrictions_on_internal_movement', 'contact_tracing', 'urban_pop_%', 'rainfall', 'relative_humidity', 'liver_disease_%', 'diarrhea_common_infectious_diseases_%', 'musculoskeletal_disorders_%', 'hospital_density', 'nbr_surgeons', 'new_cases_trend', 'new_cases_seasonality', 'lag_1', 'lag_2', 'lag_3', 'lag_4', 'lag_5'

Hyper-parameter Tuning

XGBoost Feature Selection Hyper-Parameters Tuning		
Parameter	**Values**	**Best Parameters**
learning_rate	[0.5,0.55,0.6,0.7,0.75,0.8]	0.5
max_depth	[3, 5, 7, 10]	3
min_child_weight	[1, 3, 5]	1
subsample	[0.5, 0.7]	0.7
colsample_bytree	[0.5, 0.7]	0.7
n_estimators	[10,20, 30,40,50, 60]	60

Model fitting with best parameters

We can notice an improvement in R2 score by 0.01 and RMSE by 38, and a slight deteriorating of MAE by 29.

Metric	Feature Selection Metric
r2	0.98
RMSE	787.437302
MAE	171.837341

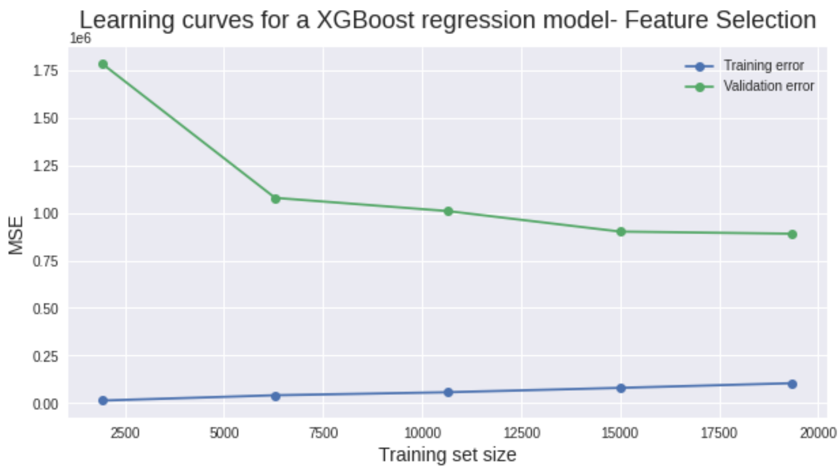
Scaling

We can notice that the scaling doesn't have a notable impact on the metrics.

	XGBoost Scaling Metric- Feature Selection				
	Standard Scaler	MinMax Scaler	Robust Scaler	Normalized scaler	Quantile Transformer
r2	0.97	0.98	0.98	0.96	0.97
RMSE	846.851931	864.275084	832.682587	897.34621	820.21171
MAE	177.833976	177.673891	172.207025	269.323567	173.761341

Learning Curve

Decreasing the numbers of features in the training data improves the model by increasing the bias and decreasing the variance. However,the gap is still wide which suggests that our model needs more training set to gives better results.



Training Size	Validation Scores	Training Scores
1934	13151.967272	1.783604e+06
6288	40955.632060	1.079596e+06
10641	56782.253795	1.010195e+06
14995	79842.917245	9.022042e+05
19349	104455.276845	8.908389e+05

Feedforward Neural Network Model

Model fitting

For our first fit of the model, we built the model by giving it some parameters including number of neurons, activation function, learning rate based on our intuition of the problem we are handling and its size. We defined our input and output layers and 2 hidden layers with relu activation function

```
def create_model():
  model = Sequential()
  model.add(Dense(128,input_dim = X_train.shape[1], activation='relu'))
  model.add(Dense(256,activation='relu'))
  model.add(Dense(256,activation='relu'))
  model.add(Dense(1,activation='relu'))
  opt = SGD(lr=0.2,clipnorm=1)
  model.compile(loss='mean_squared_error', optimizer=opt, metrics=['
   mean_absolute_error'])
  return model
```

Hyper-parameter Tuning

Inorder to perform GridSearch, Keras can be wrapped using KerasRegressor class. To use the wrapper, we defined a function that creates and returns our sequential model and pass it in KerasRegressor.

- Batch Size and Number of Epochs: **Batch Size** is the number of pattern shown to the network before the weights are updated. **epochs** is the number of times the entire dataset is shown to the network during training.
- Optimization Algorithm: Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate in order to reduce/minimize the losses.
- Learning rate, number of neurons, and activation function.

The below table shows all possible parameters we tuned for the Neural Network Model and the result of the best fit for each parameter.

Keras Hyper-Parameters Tuning		
Parameter	**Values**	**Best Parameters**
epochs	[10,20,30,50, 100]	100
batch_size	[60, 80, 100]	100
Optimzer	['SGD', 'Adagrad', 'Adam']	Adam
Learning rate	[0.08,0.1,0.2, 0.3,0.4,0.5,0.6,0.7,0.8]	0.08
Activation Function	['relu','linear']	relu
Neurons	[16,64,128,256]	128

Model fitting with best parameters

Based on the results from the hyper-parameter tuning, we fitted our model on the parameters and achieved the below evaluation metric.

Metric	Best Parameters Model
r2	0.95451
RMSE	962.849771
MAE	254.113479

```
def tuned_model(learn_rate=0.08,neurons=128,activation='relu'):
  model = Sequential()
  model.add(Dense(128,input_dim = X_train.shape[1], activation=activation))
  model.add(Dense(neurons,activation=activation))
  model.add(Dense(neurons,activation=activation))
  model.add(Dense(1,activation=activation))
  opt = Adam(lr=learn_rate,clipnorm=1)
  model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mean_absolute_error'])
  return model
```

Scaling

Neural networks weight are initially initialized to small random values and then get updated by the optimization algorithm used in the model architecture. Since the weights are initially small, scaling input values is a crucial step in neural network models. Feature scaling improves the convergence of the steepest descent algorithms,intuitively, gradient descent with features being on different scales, certain weights may update faster than others since the feature values X play a role in the weight updates. Also, scaling is used on data that consists of many different input features and each may have a different range of values (measurment units), this can clearly be shown in our data where we have columns of total COVID-19 cases, total COVID-19 deaths, whereas we also have % of female smokers. %male smokers, and ordinal categorical features including policies and governmental measurements such as school closure, work place closure... etc.

We initially scaled our data using Standard Scaler.The result of standardization (or Z-score normalization) is that the features will be rescaled so that they'll have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$ where $\mu = 0$ is the mean (average) and σ is the standard deviation from the mean.Standardizing the features is not only important if we are comparing measurements that have different units which is applicable in our data, but it is also a general requirement for many machine learning algorithms including neural network models.

Comparing the results below, we can notice that robust scaler had the best evaluation metrics followed by Standard Scaler, while other Scaling methods drastically failed to converge and didn't get good evaluation metrics. Standardization is done by removing the mean and scaling to unit variance. However, outliers can often influence the sample mean / variance in a negative way. In such cases, Robust scaler which removes the median and scales the data according to the IQR range gives us the best results.

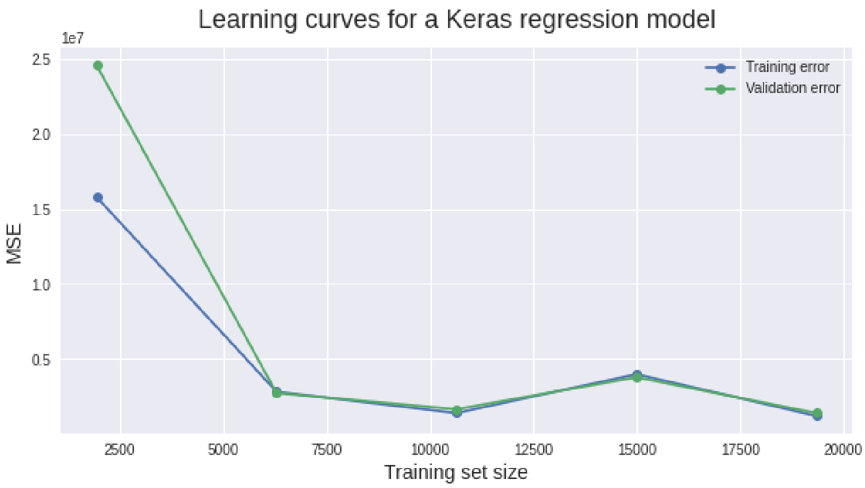
	Metric			
	Standard Scaler	**MinMax Scaler**	**Robust Scaler**	**Quantile Transformer**
r2	0.95451	0.0698	0.96322424	0.1157
RMSE	962.849771	4757.832943	901.795354	4976.818016
MAE	254.113479	1030.472066	208.449592	1151.895861

Learning Curve

As we can see in the values presented below, the trainining scores is always lower than the validation score which means that we should expect some gap between the train and validation learning curves. Observations that can be deduced from the curves:

- The plot of training scores decreases as training size increases
- The plot of validation scores decreases as training size increases
- The gap between both plots started with a high value (high variance), however as the training set increases it becomes minimal nearly zero.

- The curves didn't reach a point of stability, which suggests increasing training set size would give us better results and a chance to the model to converge to a stable error and find a global minimum of the loss function.



Training Size	Training scores	Validation scores
1934	1.580223e+07	2.459256e+07
6288	2.804098e+06	2.712180e+06
10641	1.384735e+06	1.638000e+06
14995	3.976668e+06	3.770221e+06
19349	1.193642e+06	1.383063e+06

Our model had high variance at the very first beginning of the model training, however as the train set size increases the error vanishes giving low variance low bias model.However, the curves indicates that it still have the potentials to decrease more and the model didn't reach its optimized situation (didn't converged to the minimum yet).

Keras Feature Selection

Top features are retrieved using SelectFromModel from xgboost regressor with best parameters tuned. The threshold we took is 0.001.

Features Selected	'total_cases', 'new_deaths', 'new_tests', 'total_tests', 'median_age',
	'extreme_poverty', 'female_smokers', 'school_closing',
	'international_travel_controls', 'income_support', 'debt_relief',
	'land_area_Km²', 'minimum_temperature', 'maximum_temperature',
	'relative_humidity', 'liver_disease_%',
	'diarrhea_common_infectious_diseases_%', 'hospital_density',
	'nbr_obstetricians', 'new_cases_trend', 'new_cases_seasonality',
	'lag_1', 'lag_2', 'lag_3', 'lag_4', 'lag_5'

Hyper-parameter Tuning

Keras Hyper-Parameters Tuning - Feature Selection		
Parameter	**Values**	**Best Parameters**
epochs	[10,20,30,50, 100]	100
batch_size	[60, 80, 100]	80
Optimzer	['SGD', 'Adagrad', 'Adam']	Adam
Learning rate	[0.08,0.1,0.2, 0.3,0.4,0.5]	0.1
Activation Function	['relu','linear']	relu
Neurons	[16,64,128,256]	128

Model fitting with best parameters

Based on the results from the hyper-parameter tuning, we fitted our model on the parameters and achieved the below evaluation metric. We can notice a deteriorating of R2 score by 0.00451 , improvement in RMSE by 84.712481 and in MAE by 40.18055.

Metric	Feature Selection Metric
r2	0.95
RMSE	878.13729
MAE	213.932929

Scaling

After performing feature selection in our data, we repeated the scaling exercise on the new selected features, the results shows that Standard Scaler had the best evaluation metrics among other transformations.

Keras Scaling Metric-Feature Selection				
	Standard Scaler	MinMax Scaler	Robust Scaler	Quantile Transformer
r2	0.95	0.55	0.48	0.13
RMSE	878.13729	1338.014982	1243.872488	5069.74
MAE	213.932929	364.089296	253.832856	935.299713

Learning Curve

Observations:

- The training curve decreases as training set size increases
- The validation curve decreases as training set size increases
- small gap can be observed between the training and validation curves, and it vanishes at the very end of the curve
- Validation curve had less error than training at the very first beginning.

Training Size	Training scores	Validation scores
1880	3.417421e+07	2.397141e+07
6111	7.168685e+06	8.461718e+06
10342	7.094190e+06	8.440959e+06
14573	5.607723e+06	6.671167e+06
18804	2.440699e+06	3.001822e+06

The decreasing tendency of both curves suggests that the model didn't reach to a stable point i.e. didn't yet converges and reach its minimum. This suggests that adding more training instances would give better results and could be utilized in the model convergence.

Gradient Boosting Regressor

We chose to fit a GradientBoosting regressor as a third model. Below is a code for the model fitted using KFold and cross_val score with random parameters.

```
reg = GradientBoostingRegressor(loss= 'quantile',learning_rate = 0.5,
                    max_depth = 10, n_estimators = 100,verbose = 1)
eval_set = [(X_test, y_test)]
reg.fit(X_train,y_train)
reg.predict(X_test)
reg.score(X_test, y_test)
kfold = KFold(n_splits=10, random_state=1)
results = cross_val_score(reg, X_train, y_train, cv=kfold,scoring='r2',
    verbose =1)
y_pred = reg.predict(X_test)
mse = mean_squared_error(y_pred, y_test)
```

Hyper-parameter Tuning

The below table shows the parameters we tuned for the GradientBoosting regressor model and the result of the best fit for each parameter.

GradientBoostingRegressor		
Parameters	**Values**	**Best Parameter**
learning rate	[0.1,0.3,0.5,0.7]	0.1
subsample	[0.5, 0.7]	0.7
max_depth	[3, 5, 7, 10]	10
n_estimators	[100, 200, 500]	500

Model Fitting with best parameters

Based on the results from the hyper-parameter tuning, we fitted the gradientboosting model on the parameters and achieved the below evaluation metrics.

Metric	Best Parameter Models
r2	0.97
rmse	847.568391
mae	207.055827

Scaling

We can notice that scaling doesn't have a significant impact on the metrics, and that is because Gradient Boosting Regressor is an ensemble decision tree regressor same as XGBoost scaling doesn't affect splits as they are not sensitive to transformations where a split on one scale has a corresponding split on the transformed scale.

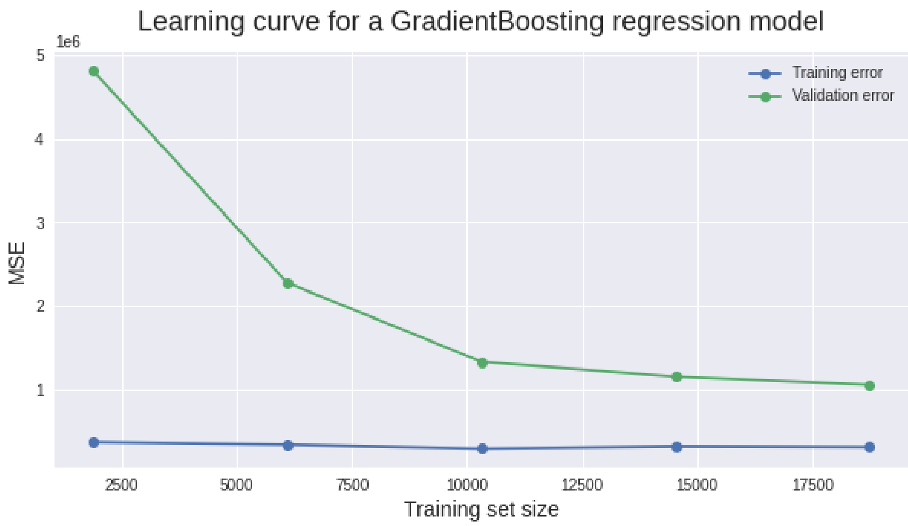
Gradient Boosting Scaling Metric				
	Standard Scaler	**MinMax Scaler**	**Robust Scaler**	**Quantile Transformer**
r2	0.97	0.97	0.97	0.97
RMSE	983.172629	992.739438	999.400662	1032.643485
MAE	218.049755	193.502709	223.597209	230.894621

Learning Curve

Same as XGBoost, our model suffers from high variance and low bias with overfitting, it can be improved by:

- Increasing the training set size
- Feature Selection.

Training Size	Validation Scores	Training scores
1874	4.81E+06	371222
6092	2.28E+06	343788
10310	1.33E+06	296517
14528	1.16E+06	323418
18747	1.06E+06	315325



Gradient Boosting Regressor Feature Selection

Top fearues are retrieved using SelectKBest with k equal 30 and f_regression as the scoring function.

Features Selected	'total_cases', 'total_deaths', 'new_deaths', 'new_tests', 'total_tests',
	'school_closing', 'workplace_closing', 'cancel_public_events',
	'restrictions_on_gathering', 'close_public_transport', 'stay_at_home',
	'restrictions_on_internal_movement', 'income_support', 'debt_relief',
	'testing_policy', 'population', 'net_change', 'land_area_Km²',
	'migrants_net', 'world_share', 'respiratory_diseases_%',
	'nbr_obstetricians', 'month', 'new_cases_trend',
	'new_cases_seasonality', 'lag_1', 'lag_2', 'lag_3', 'lag_4', 'lag_5'

We repeated the steps done previously and below is the summary of the results of each step.

Hyper-parameter Tuning

GradientBoostingRegressor with FS		
Parameters	**Values**	**Best Parameter**
learning rate	[0.01, 0.1,0.3,0.5,0.7]	0.01
subsample	[0.5, 0.7,0.9]	0.9
max_depth	[3, 5, 7, 10]	10
n_estimators	[100, 200, 500,1000]	1000

Model fitting with best Parameters and FS

After fitting the model with best parameters on the selected features we can notice an improvement in the MAE and a slight deterioration in the RMSE metric.

Metric	FS Best Parameter
r2	0.97
rmse	855.682683
mae	160.678662

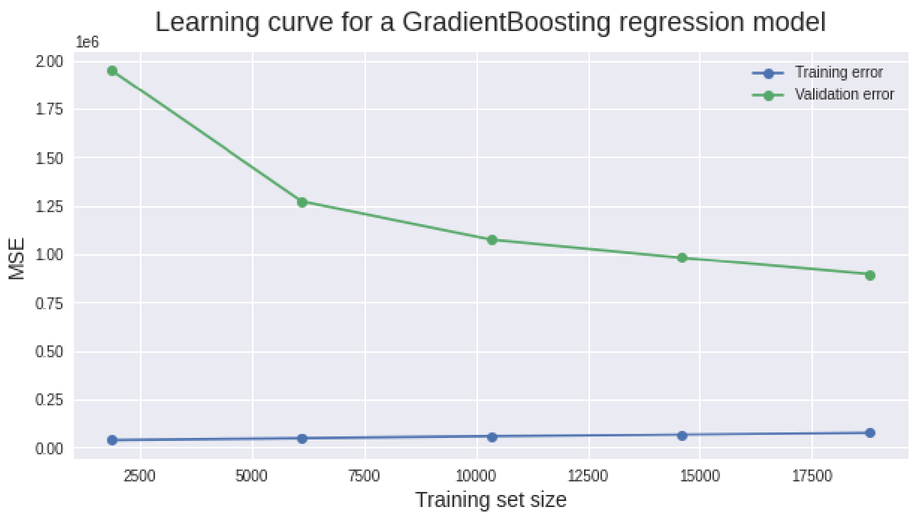
Scaling

We can notice that scaling doesn't have a significant impact on the metrics.

	Metric Feature Selection			
	Standard Scaler	**MinMax Scaler**	**Robust Scaler**	**Quantile Transformer**
R2	0.98	0.97	0.97	0.98
RMSE	718.434274	829.318182	721.660334	808.608109
MAE	150.592429	162.614821	151.868784	185.077938

Learning Curve

Decreasing the numbers of features in the training data improves the model by increasing the bias and decreasing the variance, we can notice we have a lower difference of the MSEs than before. However,the gap is still wide which suggests that our model needs more training set to gives better results.



Training Size	Validation Scores	Training scores
1880	1.95E+06	36926.40
6111	1.27E+06	46733.92
10342	1.08E+06	57432.84
14573	9.82E+05	66472.58
18804	8.95E+05	76469.88

Decision Tree Regressor Model

Since the scaling doesn't affect decision trees, we didn't show the results to avoid repetition.

Fitting the model

We fitted the model with its default parameters, KFold, and cross validation.

```
regressor = DecisionTreeRegressor(random_state=0)
kfold = KFold(n_splits=10, random_state=0)
results = cross_val_score(regressor, X_train, y_train, cv=kfold,scoring=
    'r2')
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)
```

Tuning Parameters

The below table shows the parameters we tuned for the Decision tree regressor model and the result of the best fit for each parameter.

Decision Tree Regressor Hyper-Parameters Tuning		
Parameter	**Values**	**Best Parameters**
criterion	[mse, friedman_mse, mae]	friedman_mse
min_weight_fraction_leaf	[0,1]	0

Model fitting with best parameters

Based on the results from the hyper-parameter tuning, we fitted our model on the parameters and achieved the below evaluation metric.

Metric	Best Parameters Model
r2	0.97
RMSE	1134.457563
MAE	177.168450

Learning Curve

Same as XGBoost and Gradient Boosting Regressor we can notice the gap between both training and validation curve, the model suffers also from high variance low bias, increasing training size or feature selection could help in increasing the bias and decreasing the variance.

Training Size	Training Scores	Validation Scores
1858	0.000000	4.764307e+06
6040	0.000000	2.487585e+06
10222	0.000000	1.962732e+06
14404	0.000000	1.131713e+06
18587	0.000000	9.945844e+05

Feature Selection

Top features are retrieved using SelectFromModel from decision tree regressor with best parameters tuned. The threshold we took is 0.001. We repeated the steps done previously and below is the summary of the results.

Selected Features	'new_deaths', 'yearly_change', 'nutritional_deficiencies_%',
	'new_cases_trend', 'new_cases_seasonality', 'lag_1', 'lag_4'

Hyper-parameter Tuning

Same results as before feature selection

Model fitting with best parameters after feature selection

We can notice that R2 remained the same, however RMSE improved by 112 and MAE by 20.

Metric	Feature Selection Metric
r2	0.97
RMSE	1032.032104
MAE	157.895657

Learning Curve

It looks like we had a slight improvement in the results as the validation error values are a bit lesser than before, and a small increase in the training score can be observed. But we still have the problem of over-fitting as the training error is a straight line at a very low level.

Training Size	Training Scores	Validation Scores
1858	-0.000000	2.679592e+06
6040	2.110944	2.529421e+06
10222	6.512522	1.840466e+06
14404	9.424234	9.372309e+05
18587	12.695457	1.017512e+06