Final Project

Sunghyun Park

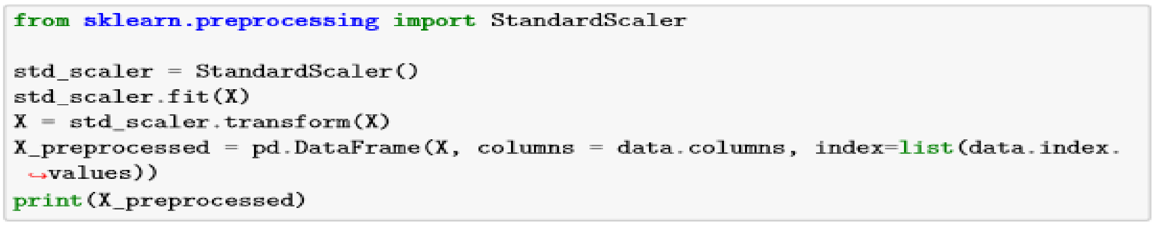
Jihoon Ban

JaeKwan Kim

“COVID-19”, one of the infectious diseases, has become a worldwide issue. The reason for that is WHO has declared this fast-spreading virus a pandemic based on the result of correlation data amongst the total population of those who have tested positive from the COVID, death cases, population of each countries, population density, and individual population rate of various ages. Shortly after announcing the state emergency, CDC (Center for Disease Control and Prevention) investigates and statistics with each state’s corona-related events. Then, corona-related statistics are published to warn how dangerous situation is going on. With these CDC data, which are total cases of positive members, deaths, population of each counties, population density, individual population rate of various ages, health outcomes, uninsured number, and unemployment number, from the CDC, we are going to make the prediction about what kind of correlation can be predicted and how much of these correlation data can be related to death by using the methods that we have learned. After then, main purpose of making the prediction is comparing between the actual number that has occurred to be death to be known how our prediction is accuracy. The methods to be used are linear regression, random forests, decision trees, and SVM.

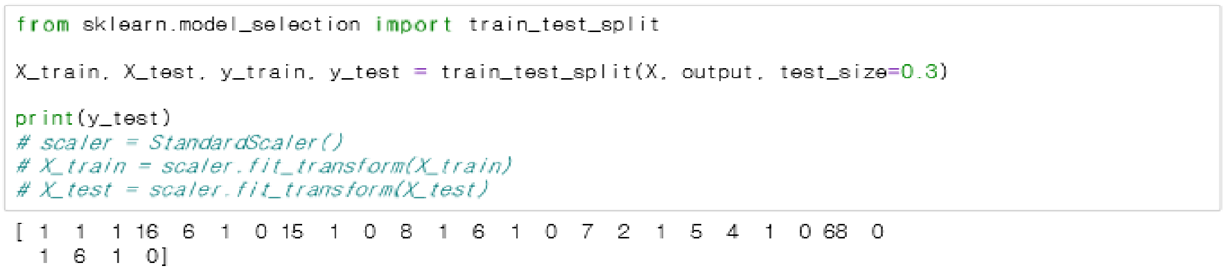
Preprocessing

Machine do not understand free text; therefore, the data should be transformed or encoded to interpret by the algorithm easily. For this reason, our team encoded our data by importing the class called StandardScaler. StandScaler class standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by standard deviation.



```python
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
std_scaler.fit(X)
X = std_scaler.transform(X)
X_preprocessed = pd.DataFrame(X, columns = data.columns, index=list(data.index.
 ↪values))
print(X_preprocessed)
```

After the preprocessing, it splits the train data and test data. The result of the ratio between train data and test data is 7 (0.7) :3 (0.3).



```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, output, test_size=0.3)

print(y_test)
# scaler = StandardScaler()
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.fit_transform(X_test)
```

```
[ 1  1  1 16  6  1  0 15  1  0  8  1  6  1  0  7  2  1  5  4  1  0 68  0
  1  6  1  0]
```

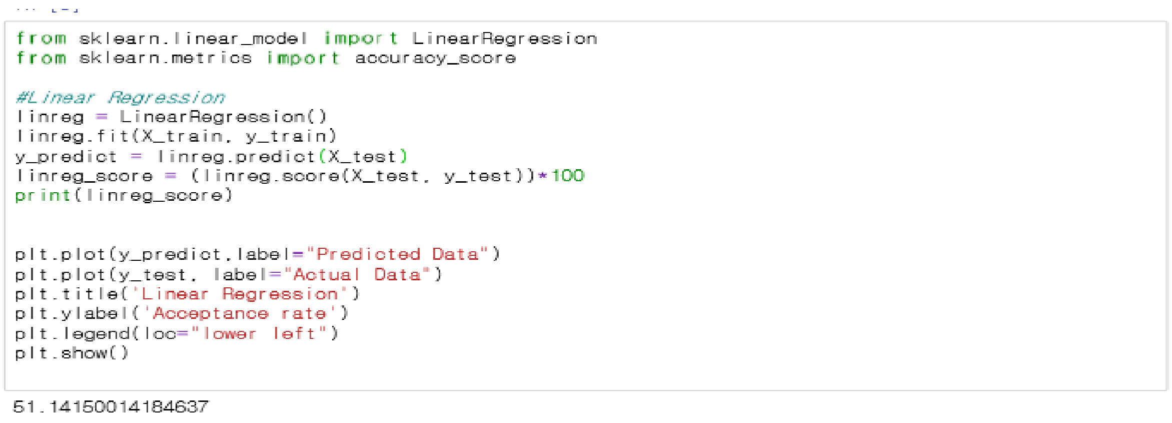
For using the model data to evaluate the performance of the model, we use the train_test_split function for splitting data arrays into four subsets: for X_train, X_test, y_train, y_test four different method that we have learned during this semester and compared the result from it.

Linear Regression

In order to perform the task to predict the COVID-19 accuracy in a simple model, Linear Regression is suited on this case. Linear regression performs the task to predict the target or dependent variable output on the basis of given independent variable input. It uses the value of intercept and slope to predict the output. Equation 1 shows the relationship between a dependent and independent a variable in linear regression model. It reveals that Linear regression finds out a linear relationship between input and output. In equation 1, beta 0 and beta 1 are two independent values which represent inctercept and slope respectively and the last word of E is the error rate. This is mostly used for predictive analysis. To make the linear regression algorithm more accurate we try to minimized the sum of residual square bewteen the predicted and actual value.

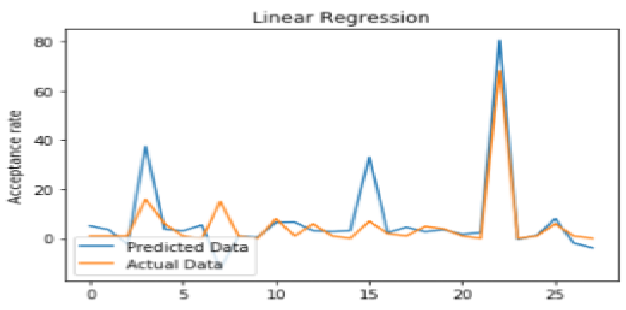
$$Y = \beta_0 + \beta_1 x + \epsilon \qquad (1)$$

We train the algorithm for each county in Indiana individually. If there are several counties in Indiana, we also train for each of them separately. Since the states are in different phases of the Covid-19, we will not be able to make good predictions with an algorithm that is trained on all data points. Besides, due to the COVID-19 confirmed cases are revealing the exponential rate, it will not have great predictions of the accuracy.



```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score

#Linear Regression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
y_predict = linreg.predict(X_test)
linreg_score = (linreg.score(X_test, y_test))*100
print(linreg_score)


plt.plot(y_predict,label="Predicted Data")
plt.plot(y_test, label="Actual Data")
plt.title('Linear Regression')
plt.ylabel('Acceptance rate')
plt.legend(loc="lower left")
plt.show()
```

```
51.14150014184637
```

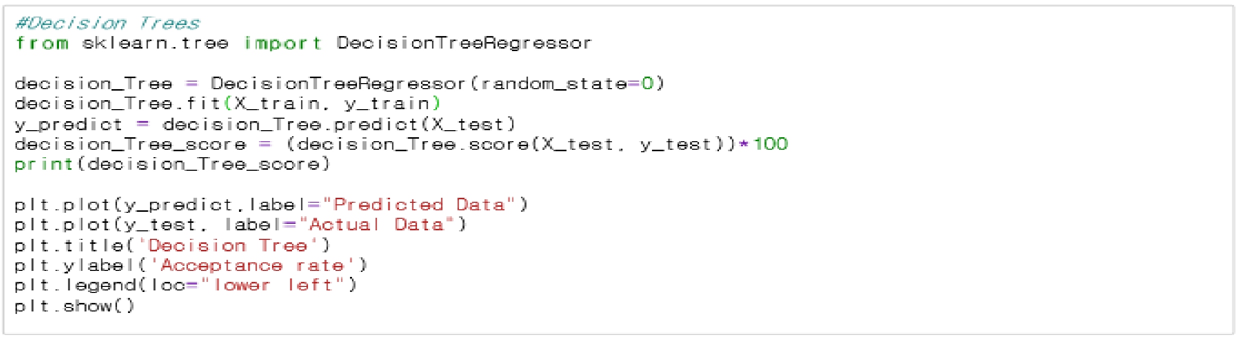


Results of the Linear Regression

For Indiana, the forecast initially looks fine. Since it can be seen that the gap between the actual data and the predicted data is widening, the accuracy for prediction represents 51 percentage which means not fitting as much as did random forest. It shows that it has to find the right place to make linear line to separate the data. In addition, because our data has many different categories to predict the COVID-19 deaths, it makes model hard to make a line to separate line. This is one of reason that the accuracy of the prediction is only 51% for this model.

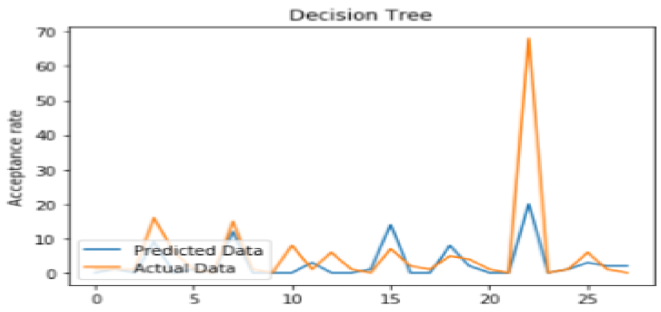
Decision Tree

For the sake of forcing the consideration of all possible outcomes from the COVID of a decision and traces each path to a conclusion, Decision tree is the appropriate algorithm. Also, decision trees assign specific values to each problem, decision path and outcome.



```
#Decision Trees
from sklearn.tree import DecisionTreeRegressor

decision_Tree = DecisionTreeRegressor(random_state=0)
decision_Tree.fit(X_train, y_train)
y_predict = decision_Tree.predict(X_test)
decision_Tree_score = (decision_Tree.score(X_test, y_test))*100
print(decision_Tree_score)

plt.plot(y_predict,label="Predicted Data")
plt.plot(y_test, label="Actual Data")
plt.title('Decision Tree')
plt.ylabel('Acceptance rate')
plt.legend(loc="lower left")
plt.show()
```

```
42.91069459757442
```

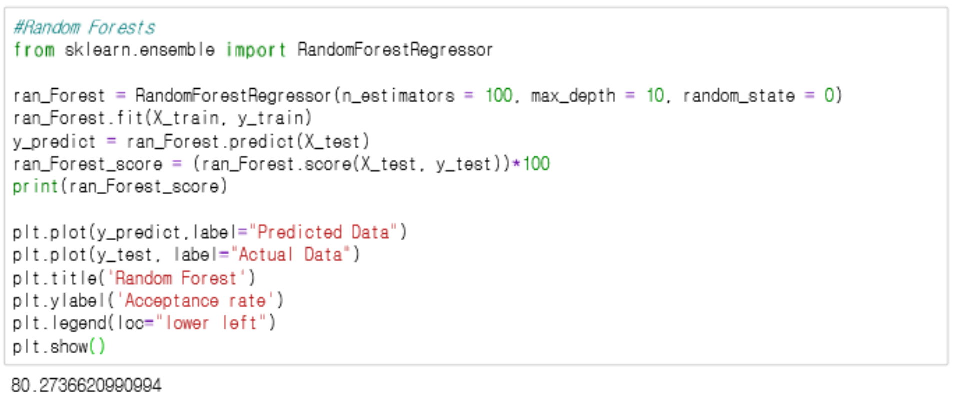


Results of the Decision Tree

From decision model what we make, it reveals the accuracy of the prediction is around 43 percentage. The decision tree makes binary questions for answer Yes or No to predict the result. At this point, because the decision tree model only has one tree and have several roots to predict, this is getting lower accurate from the prediction as one root's question is deviate from one's view. Accordingly, the decision tree has the lowest accuracy from our four different models. For that reason which the decision tree model cannot predict within several roots, this method has a risk to have overfitting the data.

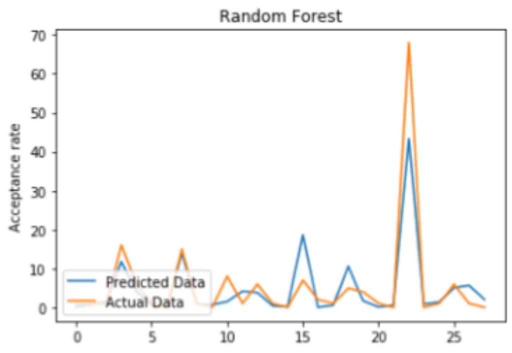
Random Forest

With using a large number of individual decision trees that operate as an ensemble, we use the function Random Forest which is a bagging technique and not a boosting technique. Because Random Forest uses bagging technique, decision trees in this algorithm are very sensitive to the data they are trained on. This brought small changes to the training set can result in significantly different tree structures. On top of that, the trees in random forests are run in parallel. Thereby, there is no interaction between these trees while building the trees. It reveals that the trees protect each other from their individual errors. Since it builds a large number of individual decision trees and trees protect each other from their individual errors, it can be more accurate and powerful predictions.



```
#Random Forests
from sklearn.ensemble import RandomForestRegressor

ran_Forest = RandomForestRegressor(n_estimators = 100, max_depth = 10, random_state = 0)
ran_Forest.fit(X_train, y_train)
y_predict = ran_Forest.predict(X_test)
ran_Forest_score = (ran_Forest.score(X_test, y_test))*100
print(ran_Forest_score)

plt.plot(y_predict,label="Predicted Data")
plt.plot(y_test, label="Actual Data")
plt.title('Random Forest')
plt.ylabel('Acceptance rate')
plt.legend(loc="lower left")
plt.show()
```

```
80.2736620990994
```

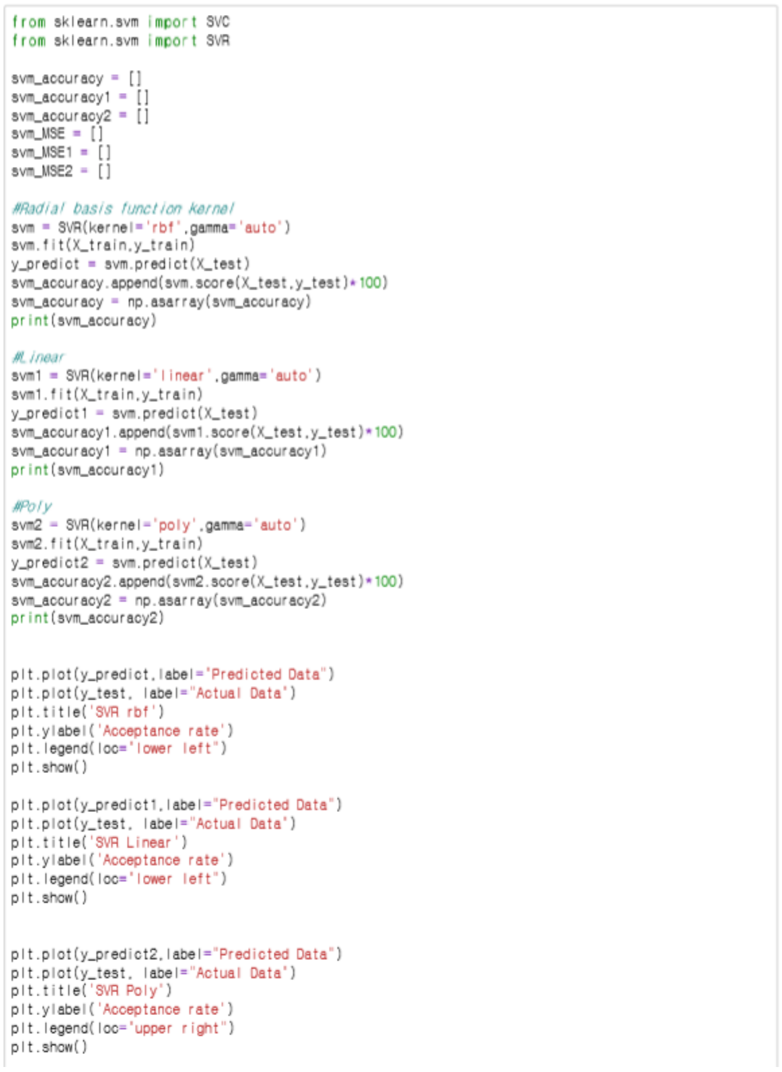


The result of the Random Forest

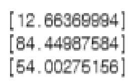
For Indiana within using the algorithm Random Forest, the forecast initially looks perfectly same as the actual data. Since it can be seen that the accuracy of the prediction is around 80 percentage which is nearby 95 percentage. One can see from the result of the random forest that since it use different decision tree to make the prediction from the data, this data becomes more accurate to forecast. Of algorithm we wrote for making the prediction, this is the highest accuracy. Moreover, it does not have any concerned about overfitting since this is using various types of decision trees.

Support Vector Machine

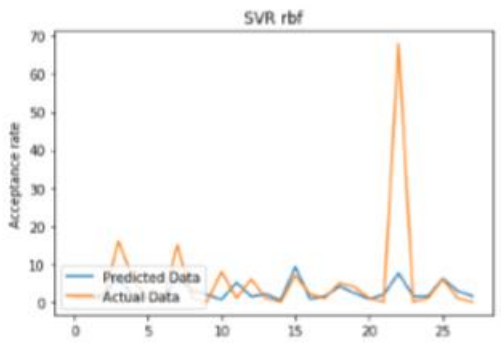
Support vector machine is supervised learning model with associated learning algorithms that analyze data used for classification and regression analysis. For the support vector machine, we used three different Kernel radial basis function, linear, and polynomial to predict the data. Radial basis functional kernel is a function whose value depends on the distance from the origin or from some point. Linear Kernel is used when the data is linearly separable, this is, it can be separated using a single line. When we use this method, it is faster than with any other Kernel. Polynomial Kernel represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.

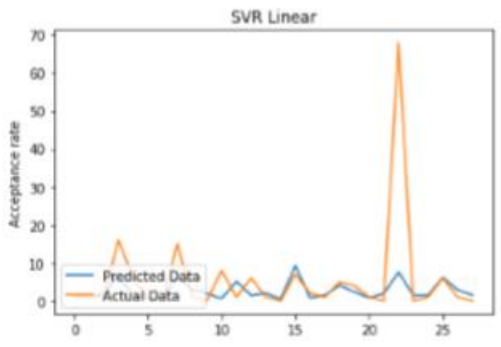


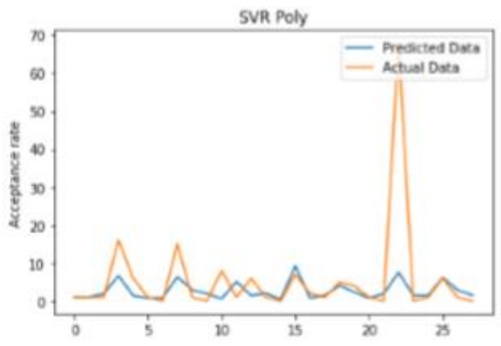
```
from sklearn.svm import SVC
from sklearn.svm import SVR

svm_accuracy = []
svm_accuracy1 = []
svm_accuracy2 = []
svm_MSE = []
svm_MSE1 = []
svm_MSE2 = []

#Radial basis function kernel
svm = SVR(kernel='rbf',gamma='auto')
svm.fit(X_train,y_train)
y_predict = svm.predict(X_test)
svm_accuracy.append(svm.score(X_test,y_test)*100)
svm_accuracy = np.asarray(svm_accuracy)
print(svm_accuracy)

#Linear
svm1 = SVR(kernel='linear',gamma='auto')
svm1.fit(X_train,y_train)
y_predict1 = svm.predict(X_test)
svm_accuracy1.append(svm1.score(X_test,y_test)*100)
svm_accuracy1 = np.asarray(svm_accuracy1)
print(svm_accuracy1)

#Poly
svm2 = SVR(kernel='poly',gamma='auto')
svm2.fit(X_train,y_train)
y_predict2 = svm.predict(X_test)
svm_accuracy2.append(svm2.score(X_test,y_test)*100)
svm_accuracy2 = np.asarray(svm_accuracy2)
print(svm_accuracy2)


plt.plot(y_predict,label='Predicted Data")
plt.plot(y_test, label="Actual Data')
plt.title('SVR rbf')
plt.ylabel('Acceptance rate')
plt.legend(loc='lower left")
plt.show()

plt.plot(y_predict1,label="Predicted Data")
plt.plot(y_test, label="Actual Data')
plt.title('SVR Linear')
plt.ylabel('Acceptance rate')
plt.legend(loc='lower left")
plt.show()


plt.plot(y_predict2,label="Predicted Data")
plt.plot(y_test, label="Actual Data')
plt.title('SVR Poly')
plt.ylabel('Acceptance rate')
plt.legend(loc='upper right")
plt.show()
```



```
[12.66369994]
[84.44987584]
[54.00275156]
```







__The result of the SVM model__

For our team's SVM model we got differently for each kernel functions. First, this data has no original point to measure the distance. Therefore, rdf function, we only get 12% of accuracy for prediction. It means that rdf function doesn't fit to this data. For the linear function we got 84% which is higher than previous linear regression model. Since SVM using kernel to make date more easily separable by linear line, so SVM model using linear function gives us more accuracy for prediction. Last, for the polynomial function, we got about 54% of accuracy, since this function represents the similarity of vectors (training samples), but our data do not have much similar training samples. This is why poly function is lower than linear function.