COVID 19 California Forecast

The responsibility of EACH TEAM MEMBER

Tommy Ly - Random Forest Regression model, Outlining Final Report and Presentation, Performance Comparison

Luke Williams - Linear Regression with Bagging

Alejandro Alatorre - Decision Tree Regressor Model and Linear Regression Model

Ryan Lee - Polynomial Regression

Victor Raj - Ridge Regression

Project description, details, goals

This dataset is of COVID 19 cases in California from January 2020 to April 2020. It shows the total number of confirmed cases and fatalities that have occurred since each day. The dataset also has longitude and latitude, although that information isn't too important as we know the data takes place in California. Our goal with the dataset will be to create a model to forecast the number of confirmed cases and fatalities on a date set in the future.

Details about the Data

Although the Kaggle dataset provides us with 3 CSV files, we will only be focusing on the train CSV file. We will be using train_test_split on the train CSV file so that we can use this as our testing data instead of the provided test CSV file, due to the testing dataset not having a label. Our data features 2 labels with continuous values. Since the data has continuous values, we must use regression and not classification. This means we will use a model on each label separately since it is not possible to predict 2 labels at the same time. We also had to modify the date column in the data so that it can be read by our models. It was modified by removing the hyphens and setting it to be an integer data type instead of a string.

The developed methods, algorithms, and tools to address the project's requirements

Algorithm #1 Random Forest Regression

This algorithm uses Random Forest Regression from the Scikit Learn ensemble library. This ensemble method will help us predict the continuous values of the labels by using many decision trees. We use the regression version of random forest instead of classifier since our labels are continuous values. Before using the Random Forest Regression model, we use the train_test_split function from Scikit Learn's library to create training and testing data.

Algorithm #2 Decision Tree Regressor

During one of our Assignments, we studied Decision Tree Classification but since our data samples are numerical data, it would be a continuous Regression model. This is why we Implemented Decision Tree Regressor. A Decision Tree Regressor uses regression to predict data. It's imported from sklearn.tree library. Decision tree regressor uses MSE and RMSE imported from metrics, to find percent error and accuracy of predicting data.

Algorithm #3 Linear Regression

In one of our Assignments we learned about the Linear Regression model. It is imported from sklearn.linear_model. The linear Regression algorithm model is used to find a relationship between one or more features, which are called independent and a continuous target variable dependent. Training a linear regression model would mean to have the best features or parameters in which the model best fits the data. But to find this we had to split the data into two subsets of training and testing data, to help us predict using linear regression Confirmed cases and Fatalities.

Algorithm #3 Revisited: Linear Regression with Bagging

Earlier, we created a linear regression model using the typical training and testing split. Now, we want to test whether this model would benefit from bagging. The predicted confirmed cases and fatalities will each get their own model. Based on the results of the

process, we will perform voting, where the average value of the models is used as the predicted value for each sample. The objective here is to achieve higher accuracy compared to our original linear regression model using voting. Ideally, this ensemble method would yield the most accurate predictions for covid-19 cases and fatalities. **Note:** because the testing data provided by Kaggle did not include any results, we had to split their original training data set into testing and training in order to evaluate accuracy.

Algorithm #4 Polynomial regression

A linear regression model requires the relation between the dependent variable and the independent variable to be linear. What if in our case the distribution of data is more complex? With Polynomial regression, we are able to generate a curve that can best capture the data.

Algorithm #5 Algorithm Ridge Regression

The Algorithmic Ridge Regression is a Linear Model from the sklearn library. Ridge Regression is beneficial when trying to find regression coefficients when the data set does not contain a large set of observations. The Algorithmic Ridge Regression model is used to eliminate multicollinearity in data models. We use Train Test Split on our training data set to randomly split our data into training and testing sets before we use the Ridge Regression model. We then get the RMSE for the Ridge Regression model to determine the accuracy for Confirmed Cases and Fatalities.

The developed codes and final results

All of our models start with this as our code:

```
import numpy as np
import pandas as pd
df_train =
pd.read_csv("https://raw.githubusercontent.com/Nbanoob/CS4661/master/ca_train.csv")
df_test =
pd.read_csv("https://raw.githubusercontent.com/Nbanoob/CS4661/master/ca_test.csv")
```

```
df_submission =
pd.read_csv("https://raw.githubusercontent.com/Nbanoob/CS4661/master/ca_su
bmission.csv")

# Modify Dates to remove dashes and to make it into an int
df_train["Date"] = df_train["Date"].apply(lambda x: x.replace("-",""))
df_train["Date"]  = df_train["Date"].astype(int)
df_test["Date"] = df_test["Date"].apply(lambda x: x.replace("-",""))
df_test["Date"]  = df_test["Date"].astype(int)
```

This code simply reads the necessary CSV files and modifies the Date column to be readable by models. Data from Date is modified by removing hyphens and setting its data type as an integer, instead of a string.

Algorithm #1 Random Forest Regression

We start by initializing the feature matrix and label vectors. This is done by using feature_cols, which consisted of lat, long, and date. We had 2 label vectors, one for confirmed cases and another for fatalities.

```
from sklearn.model_selection import train_test_split

# Initialize the feature matrix and label vector
feature_cols = ['Lat','Long','Date']
X = df_train[feature_cols]
y_case = df_train['ConfirmedCases']
y_death = df_train['Fatalities']

X_case_train, X_case_test, y_case_train, y_case_test = train_test_split(X,
y_case,test_size = 0.2, random_state = 4)
X_death_train, X_death_test, y_death_train, y_death_test =
train_test_split(X, y_death,test_size = 0.2, random_state = 4)
```

We would predict with each vector separately so that we can find the values for each label. Our random forest regressor model is imported from Scikit Learn's ensemble library and initialized with a random_state = 3 parameter.

```
from sklearn.ensemble import RandomForestRegressor
my_RandomForest = RandomForestRegressor(random_state=3)
```

We use this model to predict each of our label vectors using the test dataframe and store it in a dataframe called prediction1 and prediction2.

```
# Random Forest prediction for Confirmed Cases
my_RandomForest.fit(X_case_train,y_case_train)
prediction1 = my_RandomForest.predict(X_case_test)
prediction1 = pd.DataFrame(prediction1)
prediction1.columns = ["ConfirmedCases"]
```

```
# Random Forest prediction for Fatalities
my_RandomForest.fit(X_death_train,y_death_train)
prediction2 = my_RandomForest.predict(X_death_test)
prediction2 = pd.DataFrame(prediction2)
prediction2.columns = ["Fatalities"]
```

With our two predictions, prediction1 and prediction2, we would use Scikit Learn's library to help us calculate the mean squared error for each prediction. Once we have the mean squared error, we use NumPy to square root the mean squared error to ge the root mean square error. This is all done using this code:

```
from sklearn import metrics

mse1 = metrics.mean_squared_error(y_case_test, prediction1)
mse2 = metrics.mean_squared_error(y_death_test, prediction2)
rmse1 = np.sqrt(mse1)
rmse2 = np.sqrt(mse2)

print("Cases RMSE is: ", rmse1)
print("Fatalities RMSE is: ", rmse2)
```

This code prints out:

Confirmed Cases RMSE is: 86.12400714494434

Fatalities RMSE is: 1.1064287842563945

This shows that our Random Forest Regression model has a RMSE of 87.18 for confirmed cases and a RMSE of 1.09 for fatalities.

Algorithm #2 Decision Tree Regressor

We initialize the feature matrix and label vectors. This is done by using feature_cols, which consist of lat, long, and date. We had 2 label vectors, confirmed cases and fatalities.

```
feature_cols = ['Lat','Long','Date']
X = df_train[feature_cols]
y = df_train['ConfirmedCases']
y1 = df_train['Fatalities']
X_test = df_test[['Lat', 'Long','Date']]
print(X.head())
```

```
       Lat      Long      Date
0  36.1162 -119.6816  20200122
1  36.1162 -119.6816  20200123
2  36.1162 -119.6816  20200124
3  36.1162 -119.6816  20200125
4  36.1162 -119.6816  20200126
```

Our DecisionTreeRegressor model is imported from sklearn.tree library and is initialized with a random_state =3.

```
from sklearn.tree import DecisionTreeRegressor
DecisionTreeRegressor = DecisionTreeRegressor(random_state = 3)
```

Then the data of the train.csv file is split into training and testing prediction data with test_size of 0.2 and random_state of 4. Then we fit the model into the new training and testing data set. After, we get the predictions for each ConfirmedCases.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state =4)
X_train1, X_test1, y_train1, y_test1 = train_test_split(X,y1,test_size = 0.2, random_state =4)
```

```
] DecisionTreeRegressor.fit(X_train,y_train)
```

```
DecisionTreeRegressor(random_state=0)
```

```
] predict_ConfirmedCases = DecisionTreeRegressor.predict(X_test)
  predict_ConfirmedCases = pd.DataFrame(predict_ConfirmedCases)
  predict_ConfirmedCases.columns = ['ConfirmedCases']
```

Repeat the same process for Fatalities, which would be X_train1 and y_train1 and predict from X_test1.

```
DecisionTreeRegressor.fit(X_train1,y_train1)
```

```
DecisionTreeRegressor(random_state=3)
```

```
predict_Fatalities = DecisionTreeRegressor.predict(X_test1)
predict_Fatalities = pd.DataFrame(predict_Fatalities)
predict_Fatalities.columns = ['Fatalities']
```

Then we calculate the Root Mean Square Error (rmse) value for this DecisionTreeRegressor model,
Importing metrics from sklearn would help us get the rmse easily by just getting the data from y_test and using DecisionTreeRegressor model of predict_ConfirmedCases to find the rmse value.

```
from sklearn import metrics
mse=metrics.mean_squared_error(y_test,predict_ConfirmedCases)
rmse = np.sqrt(mse)
print("Confirmed Cases RMSE is: ", rmse)
```

The same process would be for Fatalities, but it would be from y_test1 and predict_Fatalities which would have the predictions of only Fatalities.

```
from sklearn import metrics
mse1=metrics.mean_squared_error(y_test1,predict_Fatalities)
rmse1 = np.sqrt(mse1)
print("Fatalitites Cases RMSE is: ", rmse1)
```

After applying the rmse to both ConfirmedCases and Fatalities, the output is,

```
Confirmed Cases RMSE is:  83.06635438472806
Fatalitites Cases RMSE is:  0.970725343394151
```

Meaning that for Fatalities the rmse value is very small, indicating that DecisionTreeRegressor can be a good model for prediction.

Algorithm #3 Linear Regression

We initialize the feature matrix and label vectors. This is done by using feature_cols, which consist of lat, long, and date. We had 2 label vectors, confirmed cases and fatalities.

```
feature_cols = ['Lat','Long','Date']
X = df_train[feature_cols]
y = df_train['ConfirmedCases']
y1 = df_train['Fatalities']
X_test = df_test[['Lat', 'Long','Date']]
print(X.head())
```

```
       Lat      Long      Date
0  36.1162 -119.6816  20200122
1  36.1162 -119.6816  20200123
2  36.1162 -119.6816  20200124
3  36.1162 -119.6816  20200125
4  36.1162 -119.6816  20200126
```

Then we divide into two subsets our train.csv file for training and testing data.
From sklearn.model we import train_test_split with test_size=0.2 and random_state=4.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state =4)
```

Then we call in Linear Regression model from sklearn.linear_model library and import LinearRegression. We fit our linear regression model with X_train and y_train.

```
from sklearn.linear_model import LinearRegression
my_linearReg = LinearRegression()
my_linearReg.fit(X_train,y_train)
```

Then we have to predict using the Linear Regression model with X_test data import metrics from sklearn library to help out on finding the value for rmse. In this case the rmse value using Linear Regression for ConfirmedCases would be 300.51. As we can see this is a very large error value, which indicates that LinearRegressionModel for ConfirmedCases data is not well accurate.

```
y_prediction = my_linearReg.predict(X_test)
```

```
from sklearn import metrics
mse=metrics.mean_squared_error(y_test,y_prediction)
rmse = np.sqrt(mse)
print(rmse)
```

```
300.5175754923182
```

We repeat the same process for Fatalities, we have to train split test again but now with y1 which would be fatalities. We fit linear regression model with new subset data X_train1 and y_train1. Have the linearRegression model predict X_test1 of the new subset data and calculate the rmse value for Fatalities. The value came out to be 5.29 which is somewhat small which can be one of the possible models to have a good prediction on Fatalities.

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X,y1,test_size = 0.2, random_state =4)
my_linearReg = LinearRegression()
my_linearReg.fit(X_train1,y_train1)
y_prediction2 = my_linearReg.predict(X_test1)
mse=metrics.mean_squared_error(y_test1,y_prediction2)
rmse1 = np.sqrt(mse)
print("Fatalities RMSE Value is: ", rmse1)
```

```
Fatalities RMSE Value is:  5.296290625811483
```

Algorithm #3 Revisited: Linear Regression with Bagging

Using Assignment 4 as a reference, we created a loop to train two models, one for cases and the other for fatalities. The initial setup is nearly the same as regular linear regression, where we sanitize the dates and select longitude, latitude, and date as features. In each loop, the training data is resampled with replacement using 80% of the training data. The predictions for each sample are then added to two arrays of results. As mentioned before, two “combined” arrays are then populated to reflect the outcome of the voting, where the value for each sample is the average of all 20 models. As suggested by Dr Pourhomayoun, we also tried the mode, but found the confirmed case prediction was nearly identical in accuracy, and the fatalities were slightly less accurate,

so we proceeded with using the average. Both the predicted cases and predicted fatalities are evaluated using the root mean square error.
We found that the RMSE barely changed when increasing the number of models from 10 to 20.

Using 20 models and average for voting

```
Cases RMSE is:  377.0628184232331
Fatalities RMSE is:  5.650506621703871
```

Using 20 models and mode for voting

```
Cases RMSE is:  375.82719982794896
Fatalities RMSE is:  8.693126544015074
Cases Mean Absolute Error is:  106.88663988589522
Fatalities Mean Absolute Error is:  5.888212022867028
```

10 models and mean for voting

```
Cases RMSE is:  377.60189618161917
Fatalities RMSE is:  6.022932503469911
```

In the case of covid prediction, this level of accuracy isn't remarkable, and provides more of an estimation rather than a reliable prediction. Ultimately, using bagging with linear regression did not improve accuracy over standard linear regression (see comparison section for a detailed comparison of all models). We suspect that the small dataset (63 samples) greatly limited the effectiveness of this ensemble method, and would likely be more accurate than our prior regression model had it been larger.

Algorithm #4: Polynomial Regression

First the dataset is split twice for two models one for Confirmed Cases and another for Fatalities both with a *test_size = .2* and a *random _state=4*.

```
feature_cols = ['Lat','Long','Date']
X = df_train[feature_cols]
y = df_train['ConfirmedCases']
y1 = df_train['Fatalities']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
X2_train, X2_test, y2_train, y2_test = train_test_split(X, y1, test_size=0.2, random_state=4)
```

Next we preprocess the data with *PolynomialFeatures* from *sklearn.preprocessing*. We experimented with different degrees for our polynomial feature matrix where a degree of 1 would just give us a linear model, 2 would give us a quadratic model and, 3 would give us a cubic model, from experimentation we concluded that with a maximum degree of 2 witch gives us a quadratic model would provide the best final results for our case. After the *PolynomialFeatures* variable is initialized with maximum degree of 2 we

fit_transform ‘X_train’ to the model where X_train holds the training feature data and fit_transform first computes the number of output features and transforms it to polynomial features. We then use the preprocessed data to fit into a LinearRegression model.

```
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(X_train)
pol_reg = LinearRegression()
pol_reg.fit(X_poly, y_train)
prediction_pr_cc = pol_reg.predict(poly_reg.fit_transform(X_test))
prediction_pr_cc = pd.DataFrame(prediction_pr_cc)
prediction_pr_cc.columns = ["ConfirmedCases"]
```

After fitting and predicting both models(*ConfirmedCases* and *Fatalities*), they are then evaluated with the use of metrics from SKLearn’s library to find the MSE and RMSE values for both predictions.

```
Confirmed Cases RMSE: 240.80163330608266

Fatalities RMSE: 4.229080118398884
```

WIth Polynomial regression we can see that there is a slight improvement in accuracy over the past Linear Regression model but as stated before due to the small data set a greater accuracy is difficult to achieve.

Algorithm #5 Algorithm Ridge Regression

Just as we did in the previous algorithms, we begin by initializing the feature matrix, as well as the label vectors. We create a python list called ‘feature_cols’, and initialize the list with the features ‘lat’, ‘long, and ‘date’ from our data set. We create variable ‘X’ and initialize it using ‘feature_cols’ to select the features from our data frame. We then create two label vectors, ‘y_Cases’ and ‘y_Fatalities’, and initialize each label vector to their corresponding labels from the data frame. Next, we use the SKLearn library’s test_train_split function to create our training and testing sets. We create two separate training and testing sets, where the first set is to be used to predict Confirmed Cases and the second is to be used to predict Fatalities. The ‘test_size’ is 0.2 and ‘random_state’ is 4 for the test_train_split in both sets.We import the Algorithmic Ridge

Regression from SKLearn.linear library. Then we create and initialize our ridge regression variable, ‘my_ridgereg’, with alpha=0.5 as the parameter. Alpha refers to the regularization strength, which reduces the variance of the estimations.
Then we use ‘my_ridgereg.fit(X,train, y_train)’ to train the model for the first training set for Confirmed Cases. Followed by ‘my_ridgereg.predict(X_test) to get the predictions for the Confirmed Cases. Then we repeat the same steps for the second training and testing sets. We train the model for Fatalities, and get the predictions.
We use metrics from SKLearn’s library to obtain the mse and rmse values for both predictions. The Confirmed Cases RMSE value for Ridge Regression is 300.51 The Fatalities RMSE value for Ridge Regression is 5.2962.

```
from sklearn import metrics
mse = metrics.mean_squared_error(y_test, y_Cases_predict)
rmse = np.sqrt(mse)
print('Confirmed Cases RMSE: ',rmse)
```

```
Confirmed Cases RMSE:  300.51752424083503
```

```
mse2 = metrics.mean_squared_error(y2_test, y_Fatalities_predict)
rmse2 = np.sqrt(mse2)
print('Fatalities RMSE: ',rmse2)
```

```
Fatalities RMSE:  5.296289545103671
```

Performance Comparison

By using the root mean squared error (RMSE) that we calculated in each of our models, we are able to compare the performance of our models with each other to see which model is the most effective.
This table showcases the RMSEs of our models for Confirmed Cases:

Model	Root Mean Square Error
Decision Tree Regression	83.06635438472806
Random Forest Regression	86.12400714494434
Polynomial Regression	240.8057571128469
Algorithm Ridge Regression	300.5175754923182
Linear Regression	300.5175754923182

Linear Regression with Bagging	377.0628184232331

This table showcases the RMSEs of our models for Fatalities:

Model	Root Mean Square Error
Decision Tree Regression	0.970725343394151
Random Forest Regression	1.1064287842563945
Polynomial Regression	4.229080118398884
Algorithm Ridge Regression	5.296290625811483
Linear Regression	5.296290625811483
Linear Regression with Bagging	5.650506621703871

As you can see, our **decision tree regression model has the best RMSE** for both Confirmed Cases and Fatalities, while our **linear regression with bagging model has the worse RMSE**. Our decision tree regression and random forest regression model has really low RMSEs compared to the other models and their RMSEs are very close to each other. Also, our algorithm ridge regression and linear regression models have the exact same RMSE, showing that their performance has no difference from each other.

Google Colaboratory with code and models:

https://colab.research.google.com/drive/1W6rtN0xexpcWmOUQ7PA6A3ebwAuXe9OT?usp=sharing