TJ NMLO Contest 3 Regression Solution

Edward Zhang, Sauman Das (Super Sophomores)

August 30, 2020

Abstract

In this paper, we outline our strategy for solving the third contest presented by the TJ National Machine Learning Open (NMLO). The third contest of the TJ NMLO requires competitors to predict the number of COVID-19 cases in a U.S. county given demographic data such as population, education, and income. We achieved a Mean Square Error (mse) of 2154.994 using random forest classifier. Next, we go over experimental data analysis (EDA), preprocessing, modeling, and testing.

1 Data

We looked at the data to understand what we were working with and to see which variables could be important which we could drop/engineer. According to the description, there are 4000 rows (regions of COVID-19) and 4 columns, details about each case. The column features

include education (ed), income (inc), population (pop), and cases. We start to explore the data by looking at the raw data and some datatypes.

	id	ed	inc	pop	cases
0	0	27.7	59338.0	55869.0	789
1	3	11.5	46064.0	22394.0	255
2	4	12.6	50412.0	57826.0	440
3	5	13.3	29267.0	10101.0	393
4	6	16.1	37365.0	19448.0	672

Table 1: First five rows of the data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1947 entries, 0 to 1946
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      1947 non-null   int64
 1   ed      1947 non-null   float64
 2   inc     1947 non-null   float64
 3   pop     1947 non-null   float64
 4   cases   1947 non-null   int64
dtypes: float64(3), int64(2)
memory usage: 76.2 KB
```

Figure 1: Datatypes

1.1 Data Cleaning

The first thing we did was to check for missing values and the type of each variable of each case analysis. F**igure 1** found that there were **0 missing values** and the variable types were either **int 64** or **floats .** Thus, we did not have to any data imputation nor one hot encoding since there were no categorical variables. Next, we dropped all duplicate values in the dataset since there is no effect with training.

2 Experimental Data Analysis

We started off by exploring the distribution of cases. We used matplotlib to graph the values.

```
plt.figure(figsize=(15,5))
plt.plot(train.cases,linewidth=2,color=next(color_cycle))
plt.title('Distribution Plot for Covid Cases')
plt.ylabel('# of people');
```

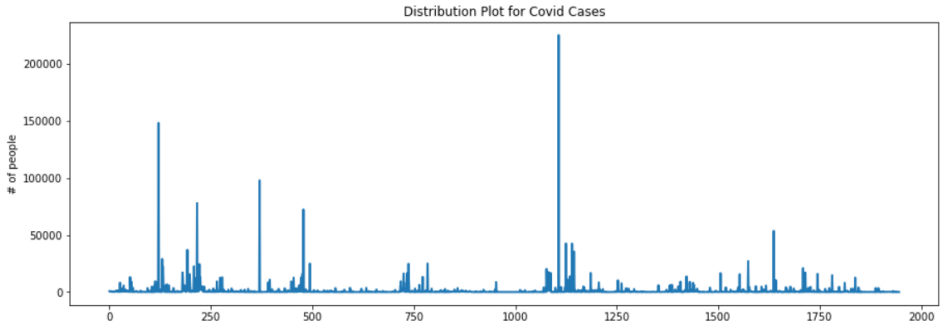

Figure 2: Distribution of Number COVID Cases According to Index Value

The data looks a bit hard to follow, so we rearranged the values from least to greatest, then plotted the data again.

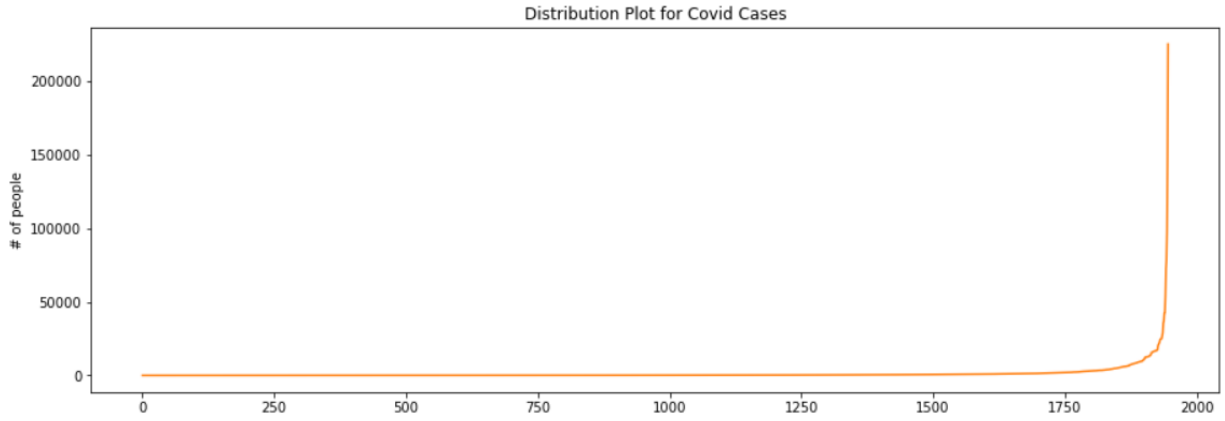

Figure 3: Distribution of Number COVID Cases (Ordered)

Figure 3 shows the plot of COVID-19 cases, we can see that this plot is not linear at all. This means the data is not linear. We were hoping to use Linear Regression to solve this problem. Because of this, we needed to transform the data to make it follow a linear distribution. Before we did this, we looked at the other variables.

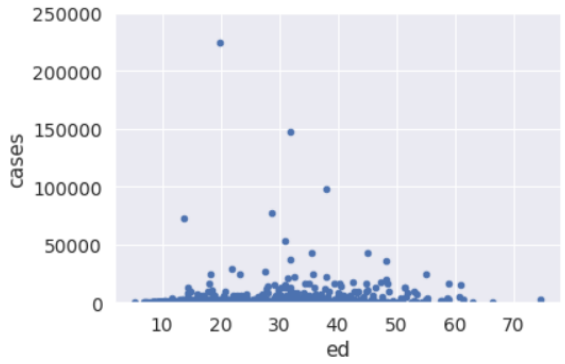

Figure 4: Distribution Plot for Education

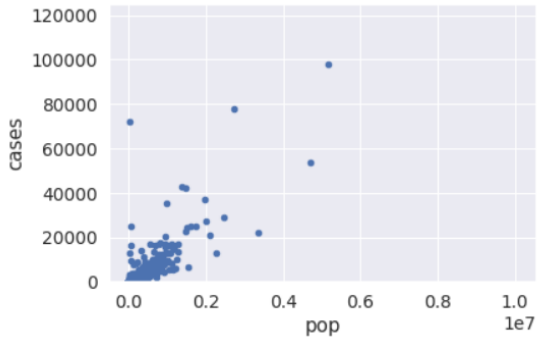

Figure 5: Distribution Plot for Population

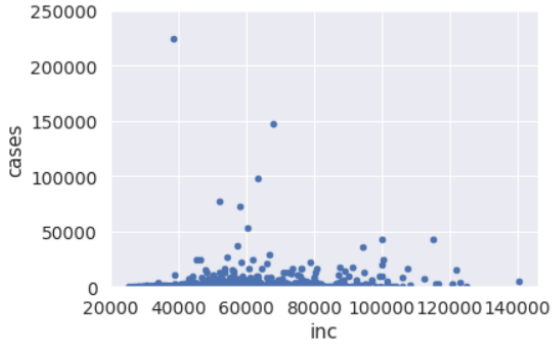

Figure 6: Distribution Plot for Income

We can see that none of the variables follow any sort of distribution. Population (pop) looks a bit linear, but we needed to investigate that a bit more. Next, we looked at the correlations of the data

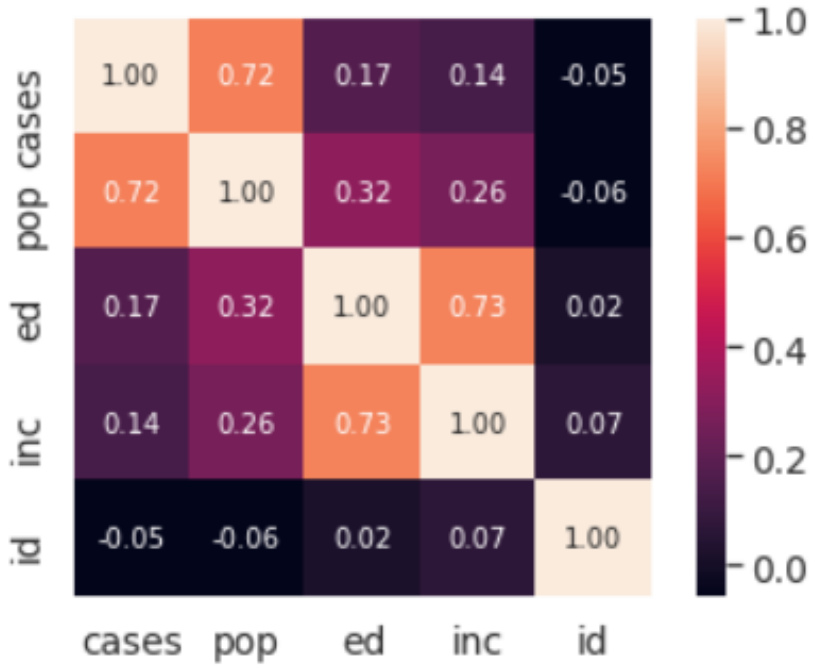

Figure 7: Correlation Heatmap

3 Preprocessing

Before modeling, we intended on normalizing our data, so that it could be linear. The purpose of this is to make modeling more effective. The model would fit straight across without missing portions of the data. Our first step to achieving this was to scale the data. We use Standard Scaler from Sklearn.

```
from sklearn.preprocessing import StandardScaler
```

```
cases_scaled = StandardScaler().fit_transform(train['cases'][:,np.newaxis]);
low_range = cases_scaled[cases_scaled[:,0].argsort()][:10]
high_range= cases_scaled[cases_scaled[:,0].argsort()][-10:]
print('outer range (low) of the distribution:')
print(low_range)
print('\nouter range (high) of the distribution:')
print(high_range)
```

```
outer range (low) of the distribution:
[[-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]
 [-0.17385952]]

outer range (high) of the distribution:
[[ 4.44529305]
 [ 4.63429509]
 [ 5.34666736]
 [ 5.36920171]
 [ 6.80202215]
 [ 9.24107329]
 [ 9.96868556]
 [12.56391414]
 [19.10408813]
 [29.13383117]]
```

Moving on, we took all the features one by one, and started to normalize the distributions. We essentially logged all of the data using np.log(). Here you can see the before and after of the distributions. We started off with cases.

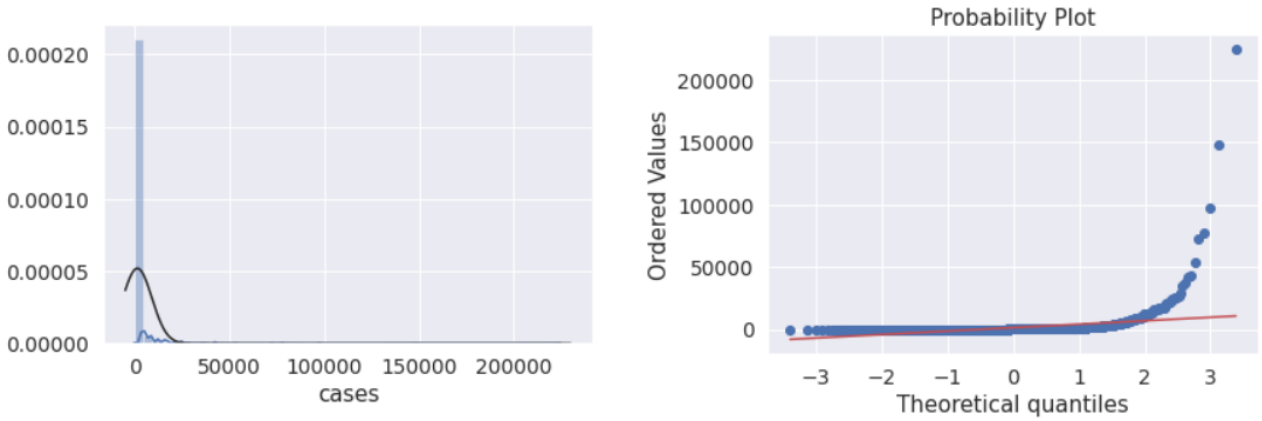

Figure 8: Cases Distribution Before Logarithmic Transformation

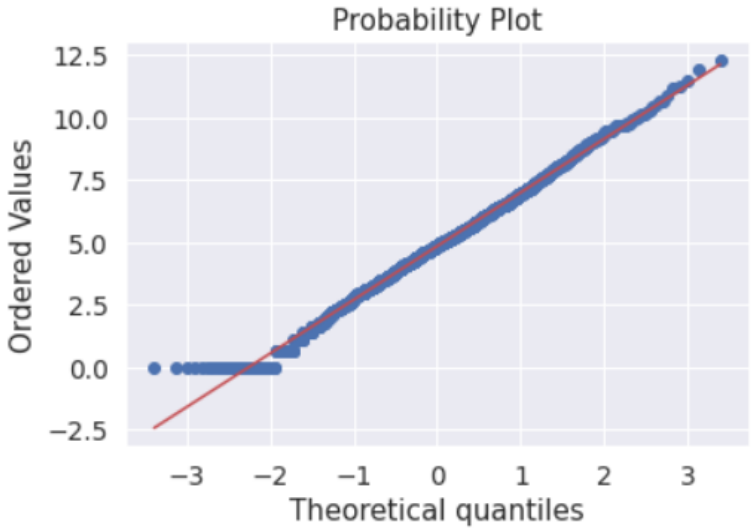

Figure 9: Case Distribution After Logarithmic Transformation

You can see after logging the parameter, the line fits better linearly. We went and did this to every single feature.

```
#transformed histogram and normal probability plot
sns.distplot(train['ed'], fit=norm);
fig = plt.figure()
res = stats.probplot(train['ed'], plot=plt)
train['ed'] = np.log(train['ed'])
test['ed'] = np.log(test['ed'])
res = stats.probplot(train['ed'], plot=plt)
```

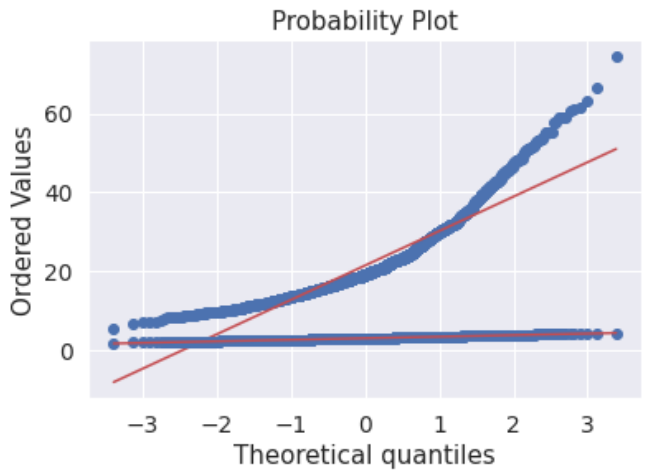

Figure 10: Education Distribution Before and After Logarithmic Transformation

Figure 11: Income Distribution Before and After Logarithmic Transformation

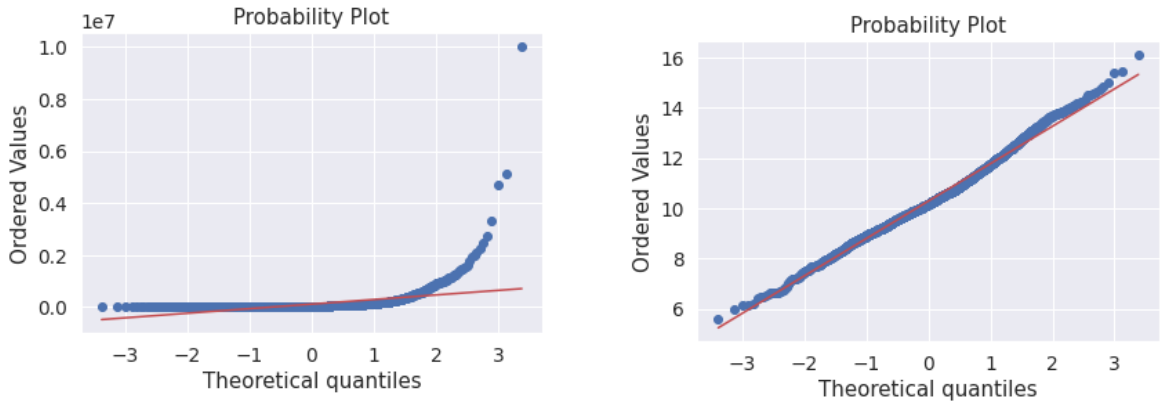

Figure 12: Population Distribution Before and After Logarithmic Transformation

Now since everything is linear, things matched up together. Take population on cases for example.

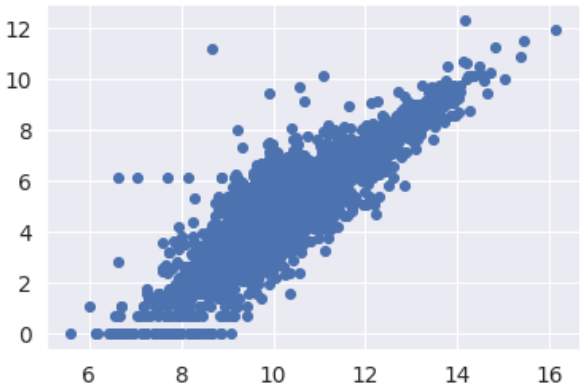

Figure 13: Population on Number of COVID-19 Cases

4 Modeling

Now that our data was prepared, we went on to modeling. We planned to try many models such as lasso regression, Gradient Boost, and more. Our best scores were achieved by Gradient Boost. First imports:

```
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge, LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
```

```
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
```

```
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9, random_state=3))
```

```
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
```

```
GBoost = GradientBoostingRegressor(n_estimators=500, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5)
```

```
model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=2200,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1)
```

We created a function to test our models.

```
n_folds = 5
def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(X_train)
    rmse= np.sqrt(-cross_val_score(model, X_train.values, y_train, scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

And finally, our results.

```
score = rmsle_cv(lasso)
print(" score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
score: 1.0319 (0.0622)
```

```
score = rmsle_cv(GBoost)
GBoost.fit(X_train, y_train)
sample = GBoost.predict(test)
sample = np.exp(sample)
print(" score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
score: 1.0674 (0.0470)
```

5 Final Thoughts

We initially got 3rd place on the public leaderboard, but then we dropped to 17th place on the private leaderboard. I believe this is because we overfitted a bit with our predictions. Everyone also did better on the private leaderboard which we did not expect. Overall, we thought that this competition was great especially in our status quo of COVID-19. We thought this competition gave us a good viewpoint on how we, even as high schoolers, can make a difference to solve relevant problems.