Analysis of Machine Learning Regression Models on Predicting Coronavirus Spread

Anish Mirjankar, 5/5/2020

The 2019 spread of Coronavirus, or COVID-19, is considered one of the largest pandemics in decades. Data is being collected by local, state, and federal government agencies across the country, as well as countless private organizations, tracking testing and cases as they develop around the country. I chose to use data from the COVID Tracking Project, an organization which maintains an api of all current and historical cases collected from the entire country.

Some questions that I am looking to answer are:

1. Does this data suffice to train a model that can successfully predict the spread of COVID in an area given the historical data provided?
2. Can we predict the number of total cases in a given region based on the testing/hospitalization data, state information, and any other categorical parameters included?
3. Can this model predict certain outcomes based on the pre-processed, daily intake data provided by the API?
4. Which method of machine learning can generate the most succesful prediction/model of the coronavirus spread?

Data sources:

- the COVID tracking Project (https://covidtracking.com)
- United States Census (https://www.census.gov/data/tables/time-series/demo/popest/2010s-state-total.html)

In [1]:
```python
import pandas as pd
links = {
    'curr_state': 'https://covidtracking.com/api/v1/states/current.csv',
    'hist_state': 'https://covidtracking.com/api/v1/states/daily.csv',
    'state_info': 'https://covidtracking.com/api/v1/states/info.csv',
    'curr_us': 'https://covidtracking.com/api/v1/us/current.csv',
    'hist_us': 'https://covidtracking.com/api/v1/us/daily.csv',
    'tracker': 'https://covidtracking.com/api/v1/urls.csv',
    'state_pages': 'https://covidtracking.com/api/v1/states/screenshots.csv',
    'state_pop':'https://www2.census.gov/programs-surveys/popest/datasets/2010
-2019/counties/totals/co-est2019-alldata.csv'
}

df_hist = pd.read_csv(links['hist_state'])
df_hist
```

Out[1]:

	date	state	positive	negative	pending	hospitalizedCurrently	hospitalizedCumulative
0	20200505	AK	371.0	22321.0	NaN	13.0	NaN
1	20200505	AL	8285.0	98481.0	NaN	NaN	1107.0
2	20200505	AR	3496.0	51139.0	NaN	89.0	453.0
3	20200505	AS	0.0	83.0	NaN	NaN	NaN
4	20200505	AZ	9305.0	78955.0	NaN	728.0	1397.0
...	...	...	...	...	...	...	...
3428	20200126	WA	1.0	NaN	NaN	NaN	NaN
3429	20200125	WA	1.0	NaN	NaN	NaN	NaN
3430	20200124	WA	1.0	NaN	NaN	NaN	NaN
3431	20200123	WA	1.0	NaN	NaN	NaN	NaN
3432	20200122	WA	1.0	NaN	NaN	NaN	NaN

3433 rows × 27 columns

Data Cleanup & Pre-Processing

The data provided requires cleanup and pre-processing to be suitable for intake by a regression model. First, all categorical fields must either be dumped or converted to a integer/float.

Adding population data

This specific dataset highly requires differentiation between states, therefore the state name is converted to its index in a static list of states and their population data. The two dataframes are joined to provide each region with its associated states population data.

In [2]:
```
import numpy as np

def generate_df(old):
    states_to_abb = {"Alabama": "AL", "Alaska": "AK", "Arizona": "AZ", "Arkans
as": "AR", "California": "CA", "Colorado": "CO",
    "Connecticut": "CT", "Delaware": "DE", "Florida": "FL", "Georgia": "GA",
"Hawaii": "HI", "Idaho": "ID", "Illinois": "IL",
    "Indiana": "IN", "Iowa": "IA", "Kansas": "KS", "Kentucky": "KY", "Louisian
a": "LA", "Maine": "ME", "Maryland": "MD",
    "Massachusetts": "MA", "Michigan": "MI", "Minnesota": "MN", "Mississippi":
"MS", "Missouri": "MO", "Montana": "MT",
    "Nebraska": "NE", "Nevada": "NV", "New Hampshire": "NH", "New Jersey": "N
J", "New Mexico": "NM", "New York": "NY",
    "North Carolina": "NC", "North Dakota": "ND", "Ohio": "OH", "Oklahoma": "O
K", "Oregon": "OR", "Pennsylvania": "PA",
    "Rhode Island": "RI", "South Carolina": "SC", "South Dakota": "SD", "Tenne
ssee": "TN", "Texas": "TX", "Utah": "UT",
    "Vermont": "VT", "Virginia": "VA", "Washington": "WA", "West Virginia": "W
V", "Wisconsin": "WI", "Wyoming": "WY",
    "District of Columbia": "DC"}

    df_states = pd.read_csv(links['state_pop'], encoding="latin-1")
    df_states["statecode"] = df_states['STNAME'].apply(lambda x: states_to_abb
[x])

    df = pd.merge(old, df_states, left_on="state", right_on="statecode", how=
"left")
    df = df[df['statecode'].notna()]
    cols_to_retain = [k for k,v in df.dtypes.to_dict().items() if v in [np.flo
at64, np.int64]]
    # cleanup non-numerical columns
    df = df[cols_to_retain]
    return df
df = generate_df(df_hist)
df
```

Out[2]:

	date	positive	negative	pending	hospitalizedCurrently	hospitalizedCumulative	inIc
0	20200505	371.0	22321.0	NaN	13.0	NaN	
1	20200505	371.0	22321.0	NaN	13.0	NaN	
2	20200505	371.0	22321.0	NaN	13.0	NaN	
3	20200505	371.0	22321.0	NaN	13.0	NaN	
4	20200505	371.0	22321.0	NaN	13.0	NaN	
...	...	...	...	...	...	...	
198955	20200122	1.0	NaN	NaN	NaN	NaN	
198956	20200122	1.0	NaN	NaN	NaN	NaN	
198957	20200122	1.0	NaN	NaN	NaN	NaN	
198958	20200122	1.0	NaN	NaN	NaN	NaN	
198959	20200122	1.0	NaN	NaN	NaN	NaN	

198705 rows × 184 columns

In [3]:
```
print(list(df.columns))
```

```
['date', 'positive', 'negative', 'pending', 'hospitalizedCurrently', 'hospita
lizedCumulative', 'inIcuCurrently', 'inIcuCumulative', 'onVentilatorCurrentl
y', 'onVentilatorCumulative', 'recovered', 'death', 'hospitalized', 'total',
'totalTestResults', 'posNeg', 'fips', 'deathIncrease', 'hospitalizedIncreas
e', 'negativeIncrease', 'positiveIncrease', 'totalTestResultsIncrease', 'SUML
EV', 'REGION', 'DIVISION', 'STATE', 'COUNTY', 'CENSUS2010POP', 'ESTIMATESBASE
2010', 'POPESTIMATE2010', 'POPESTIMATE2011', 'POPESTIMATE2012', 'POPESTIMATE2
013', 'POPESTIMATE2014', 'POPESTIMATE2015', 'POPESTIMATE2016', 'POPESTIMATE20
17', 'POPESTIMATE2018', 'POPESTIMATE2019', 'NPOPCHG_2010', 'NPOPCHG_2011', 'N
POPCHG_2012', 'NPOPCHG_2013', 'NPOPCHG_2014', 'NPOPCHG_2015', 'NPOPCHG_2016',
'NPOPCHG_2017', 'NPOPCHG_2018', 'NPOPCHG_2019', 'BIRTHS2010', 'BIRTHS2011',
'BIRTHS2012', 'BIRTHS2013', 'BIRTHS2014', 'BIRTHS2015', 'BIRTHS2016', 'BIRTHS
2017', 'BIRTHS2018', 'BIRTHS2019', 'DEATHS2010', 'DEATHS2011', 'DEATHS2012',
'DEATHS2013', 'DEATHS2014', 'DEATHS2015', 'DEATHS2016', 'DEATHS2017', 'DEATHS
2018', 'DEATHS2019', 'NATURALINC2010', 'NATURALINC2011', 'NATURALINC2012', 'N
ATURALINC2013', 'NATURALINC2014', 'NATURALINC2015', 'NATURALINC2016', 'NATURA
LINC2017', 'NATURALINC2018', 'NATURALINC2019', 'INTERNATIONALMIG2010', 'INTER
NATIONALMIG2011', 'INTERNATIONALMIG2012', 'INTERNATIONALMIG2013', 'INTERNATIO
NALMIG2014', 'INTERNATIONALMIG2015', 'INTERNATIONALMIG2016', 'INTERNATIONALMI
G2017', 'INTERNATIONALMIG2018', 'INTERNATIONALMIG2019', 'DOMESTICMIG2010', 'D
OMESTICMIG2011', 'DOMESTICMIG2012', 'DOMESTICMIG2013', 'DOMESTICMIG2014', 'DO
MESTICMIG2015', 'DOMESTICMIG2016', 'DOMESTICMIG2017', 'DOMESTICMIG2018', 'DOM
ESTICMIG2019', 'NETMIG2010', 'NETMIG2011', 'NETMIG2012', 'NETMIG2013', 'NETMI
G2014', 'NETMIG2015', 'NETMIG2016', 'NETMIG2017', 'NETMIG2018', 'NETMIG2019',
'RESIDUAL2010', 'RESIDUAL2011', 'RESIDUAL2012', 'RESIDUAL2013', 'RESIDUAL201
4', 'RESIDUAL2015', 'RESIDUAL2016', 'RESIDUAL2017', 'RESIDUAL2018', 'RESIDUAL
2019', 'GQESTIMATESBASE2010', 'GQESTIMATES2010', 'GQESTIMATES2011', 'GQESTIMA
TES2012', 'GQESTIMATES2013', 'GQESTIMATES2014', 'GQESTIMATES2015', 'GQESTIMAT
ES2016', 'GQESTIMATES2017', 'GQESTIMATES2018', 'GQESTIMATES2019', 'RBIRTH201
1', 'RBIRTH2012', 'RBIRTH2013', 'RBIRTH2014', 'RBIRTH2015', 'RBIRTH2016', 'RB
IRTH2017', 'RBIRTH2018', 'RBIRTH2019', 'RDEATH2011', 'RDEATH2012', 'RDEATH201
3', 'RDEATH2014', 'RDEATH2015', 'RDEATH2016', 'RDEATH2017', 'RDEATH2018', 'RD
EATH2019', 'RNATURALINC2011', 'RNATURALINC2012', 'RNATURALINC2013', 'RNATURAL
INC2014', 'RNATURALINC2015', 'RNATURALINC2016', 'RNATURALINC2017', 'RNATURALI
NC2018', 'RNATURALINC2019', 'RINTERNATIONALMIG2011', 'RINTERNATIONALMIG2012',
'RINTERNATIONALMIG2013', 'RINTERNATIONALMIG2014', 'RINTERNATIONALMIG2015', 'R
INTERNATIONALMIG2016', 'RINTERNATIONALMIG2017', 'RINTERNATIONALMIG2018', 'RIN
TERNATIONALMIG2019', 'RDOMESTICMIG2011', 'RDOMESTICMIG2012', 'RDOMESTICMIG201
3', 'RDOMESTICMIG2014', 'RDOMESTICMIG2015', 'RDOMESTICMIG2016', 'RDOMESTICMIG
2017', 'RDOMESTICMIG2018', 'RDOMESTICMIG2019', 'RNETMIG2011', 'RNETMIG2012',
'RNETMIG2013', 'RNETMIG2014', 'RNETMIG2015', 'RNETMIG2016', 'RNETMIG2017', 'R
NETMIG2018', 'RNETMIG2019']
```

Model training

The following n-fold cross validation function is used to accurately train and test each algorithm and its validity. It will be used to compare the algorithm on a subset of metrics, including MAE, MSE, and RMSE.

In [4]:
```
import numpy as np
import pandas as pd
import datetime as dt

def n_fold_cross_validation(dataset, class_name, n_splits=5, metrics={}, **kwa
rgs):
    """
    dataset = dataset to split
    class_name = results class column name
    n_splits = number of splits to use
    metrics = list of metrics to include
    kwargs = dict of estimators to use (pass as name=MLClass)
    """
    results = {k:{m:0 for m, _ in metrics.items()} for k, _ in kwargs.items()}
    split_data = np.array_split(dataset.sample(frac=1), n_splits)
    print(results)
    for i, testing in enumerate(split_data):
        print(f"\nIteration {i+1}: ")
        # creating tr_attributes, tr_class, te_attributes, te_class = training
and testing frames
        training = pd.concat([df for j, df in enumerate(split_data) if j != i
])

        # fill unknown results - mean for numerical data and most frequent for
categorical data
        training = training.fillna(training.mean())
        testing = testing.fillna(testing.mean())

        tr_attributes, tr_class = training.drop(columns=class_name).to_numpy
(), training[class_name].to_numpy()
        te_attributes, te_class = testing.drop(columns=class_name).to_numpy(),
testing[class_name].to_numpy()

        for j, (name, model) in enumerate(kwargs.items()):
            start_time = dt.datetime.now()
            model = model.fit(tr_attributes, tr_class)
            tr_time = dt.datetime.now()

            # for seeing the results in original dataset
            #testing["prediction"] = model.predict(te_attributes)
            #print(testing)

            for k, func in metrics.items():
                results[name][k] += func(model.predict(te_attributes), te_clas
s)
            end_time = dt.datetime.now()
            print(f"Training time for {name}: {tr_time-start_time}s \nTesting
 time for {name}: {end_time-tr_time}s")

    # divide summed metrics by number of splits to find final average
    for k, v  in results.items():
        results[k] = {met_name:met_data/n_splits for met_name, met_data in v.i
tems()}

    return results
```

Model tuning

In order to properly tune each model for the use case at hand, the below implementations were created. All models can then be compared by their results from the n-fold cross validation function above. The following regression models were selected due to their individual characteristics:

- Linear Regression Model
- Decision Tree Regressor
- MLP Regressor
- Support Vector Regressor

In [5]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn import metrics

linear = LinearRegression()
dtree = DecisionTreeRegressor()
mlp = MLPRegressor()
svm = SVR()

results = n_fold_cross_validation(
    df,
    metrics={
        'mae':metrics.mean_absolute_error,
        'mse':metrics.mean_squared_error,
        'rmse': lambda x,y: np.sqrt(metrics.mean_absolute_error(x, y))
    },
    n_splits=5,
    class_name="hospitalizedIncrease",
    lin=linear,
    dtree=dtree,
)
```

```
{'lin': {'mae': 0, 'mse': 0, 'rmse': 0}, 'dtree': {'mae': 0, 'mse': 0, 'rms
e': 0}}

Iteration 1:
Training time for lin: 0:00:03.041407s
Testing time for lin: 0:00:00.056997s
Training time for dtree: 0:00:21.818264s
Testing time for dtree: 0:00:00.132025s

Iteration 2:
Training time for lin: 0:00:02.814696s
Testing time for lin: 0:00:00.048519s
Training time for dtree: 0:00:22.342130s
Testing time for dtree: 0:00:00.146007s

Iteration 3:
Training time for lin: 0:00:02.955501s
Testing time for lin: 0:00:00.059063s
Training time for dtree: 0:00:25.652770s
Testing time for dtree: 0:00:00.169519s

Iteration 4:
Training time for lin: 0:00:03.031510s
Testing time for lin: 0:00:00.054002s
Training time for dtree: 0:00:25.211007s
Testing time for dtree: 0:00:00.144831s

Iteration 5:
Training time for lin: 0:00:03.623542s
Testing time for lin: 0:00:00.071005s
Training time for dtree: 0:00:26.943555s
Testing time for dtree: 0:00:00.151003s
```

Analysis

Analysis was performed on each model to determine the validity and accuracy of the model, as well as the training time and prediction time.

Runtime

- Linear Regression
- MLP - Eliminated for runtime constraints.
- Decision Tree Regression
- SVM Regression - Eliminated for runtime constraints.

Model metrics

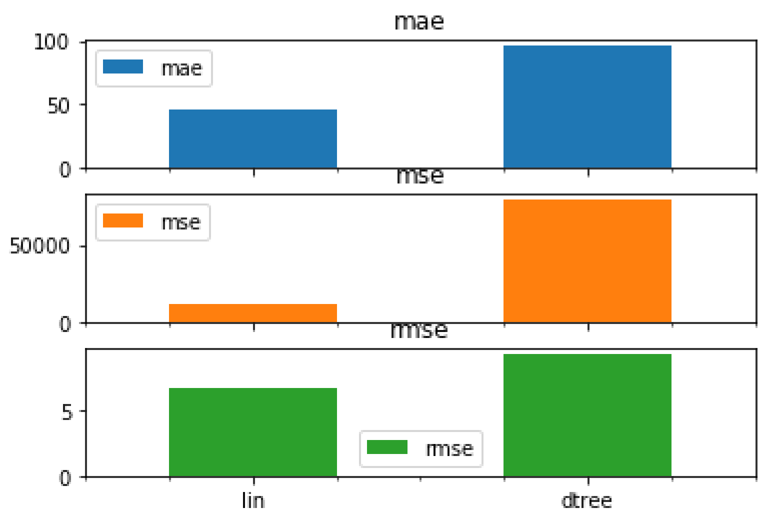
The linear regression performed the calculation with an RMSE of 6.83, compared to 6.98 of the decision tree implementation. These two proved the most comparable in terms of accuracy and runtime.

In [6]:
```
import matplotlib.pyplot as plt

resdf = pd.DataFrame.from_dict(results, orient="index")

axes = resdf.plot.bar(rot=0, subplots=True)
_ = [print(k,":",v) for k,v in results.items()]
```

```
lin : {'mae': 46.654293021122356, 'mse': 12193.496818283751, 'rmse': 6.830164
991274373}
dtree : {'mae': 95.49537387374498, 'mse': 79046.26628628402, 'rmse': 9.419645
386024296}
```



Conclusion

The questions proposed in the introduction are answered below:

1. Does this data suffice to train a model that can successfully predict the spread of COVID in an area given the historical data provided?

```
 Yes, several models were used, several of which trained to make a prediction w
ithin the scope of the project.
 Each model was measured by its Mean Absolute Error, Mean Squared Error, and Ro
ot Mean Squared Error.
```

2. Can we predict the number of total cases in a given region based on the testing/hospitalization data, state information, and any other categorical parameters included?

```
 Yes, we can accurately predict the number of cases or the increase in a region
given state population and growth
 and the given testing and hospitalization data.
```

3. Can this model predict certain outcomes based on the pre-processed, daily intake data provided by the API?

```
 Yes, the linear and decision tree model can't generate a prediction based on t
he value from current state data,
 but significant preprocessing is required to prepare the input data for predic
tion.
```

4. Which method of machine learning can generate the most succesful prediction/model of the coronavirus spread?

```
 A linear regression model most successfuly models the data with the least erro
r, and quickest runtime.
```

In [7]:
```
test_df = pd.read_csv(links['curr_state'])
clean_df = generate_df(test_df).fillna(test_df.mean())
print(df.columns)
print(clean_df.columns)
clean_df['lin_pred'] = linear.predict(clean_df.to_numpy())
clean_df['dtree_pred'] = dtree.predict(clean_df.to_numpy())
```

```
Index(['date', 'positive', 'negative', 'pending', 'hospitalizedCurrently',
       'hospitalizedCumulative', 'inIcuCurrently', 'inIcuCumulative',
       'onVentilatorCurrently', 'onVentilatorCumulative',
       ...
       'RDOMESTICMIG2019', 'RNETMIG2011', 'RNETMIG2012', 'RNETMIG2013',
       'RNETMIG2014', 'RNETMIG2015', 'RNETMIG2016', 'RNETMIG2017',
       'RNETMIG2018', 'RNETMIG2019'],
      dtype='object', length=184)
Index(['positive', 'positiveScore', 'negativeScore', 'negativeRegularScore',
       'commercialScore', 'score', 'negative', 'pending',
       'hospitalizedCurrently', 'hospitalizedCumulative',
       ...
       'RDOMESTICMIG2019', 'RNETMIG2011', 'RNETMIG2012', 'RNETMIG2013',
       'RNETMIG2014', 'RNETMIG2015', 'RNETMIG2016', 'RNETMIG2017',
       'RNETMIG2018', 'RNETMIG2019'],
      dtype='object', length=183)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-7-5a455b82719a> in <module>
      4 print(clean_df.columns)
      5 clean_df['lin_pred'] = linear.predict(clean_df.to_numpy())
----> 6 clean_df['dtree_pred'] = dtree.predict(clean_df.to_numpy())

~\jupyter\lib\site-packages\sklearn\tree\_classes.py in predict(self, X, chec
k_input)
    417         """
    418         check_is_fitted(self)
--> 419         X = self._validate_X_predict(X, check_input)
    420         proba = self.tree_.predict(X)
    421         n_samples = X.shape[0]

~\jupyter\lib\site-packages\sklearn\tree\_classes.py in _validate_X_predict(s
elf, X, check_input)
    389                              "match the input. Model n_features is %s
and "
    390                              "input n_features is %s "
--> 391                              % (self.n_features_, n_features))
    392 
    393         return X

ValueError: Number of features of the model must match the input. Model n_fea
tures is 183 and input n_features is 184
```