RandomForest_casestudy_covid19

September 9, 2020

0.1 Random Forest

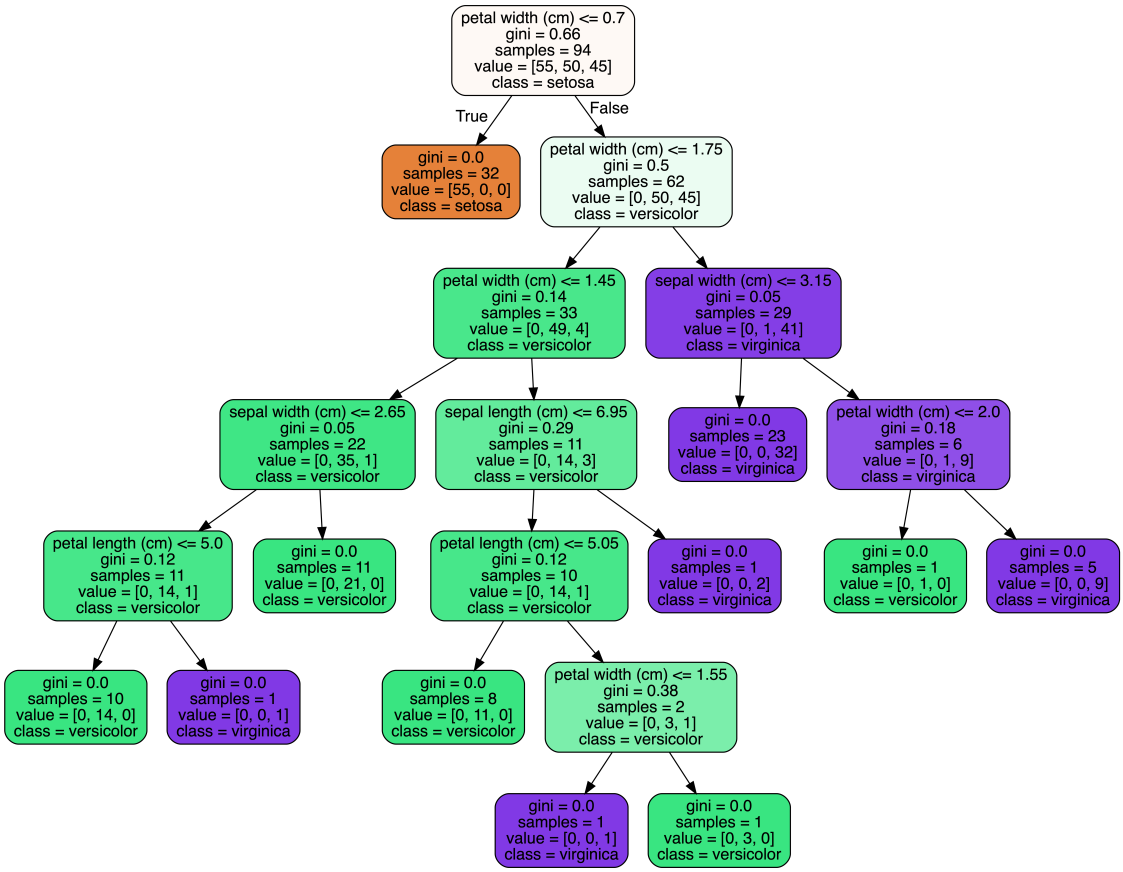
Random Forest is an ensemble of Decision Trees. With a few exceptions, a `RandomForestClassifier` has all the hyperparameters of a `DecisionTreeClassifier` (to control how trees are grown), plus all the hyperparameters of a `BaggingClassifier` to control the ensemble itself.

The Random Forest algorithm introduces extra randomness when growing trees; instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This results in a greater tree diversity, which (once again) trades a higher bias for a lower variance, generally yielding an overall better model. The following `BaggingClassifier` is roughly equivalent to the previous `RandomForestClassifier`. Run the cell below to visualize a single estimator from a random forest model, using the Iris dataset to classify the data into the appropriate species.

[309]:
```
from sklearn.datasets import load_iris
iris = load_iris()

# Model (can also use single decision tree)
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=10)

# Train
model.fit(iris.data, iris.target)
# Extract single tree
estimator = model.estimators_[5]

from sklearn.tree import export_graphviz
# Export as dot file
export_graphviz(estimator, out_file='tree.dot',
                feature_names = iris.feature_names,
                class_names = iris.target_names,
                rounded = True, proportion = False,
                precision = 2, filled = True)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])
```

```
# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

[309]:



Notice how each split seperates the data into buckets of similar observations. This is a single tree and a relatively simple classification dataset, but the same method is used in a more complex dataset with greater depth to the trees.

0.2 Coronavirus

Coronavirus disease (COVID-19) is an infectious disease caused by a new virus. The disease causes respiratory illness (like the flu) with symptoms such as a cough, fever, and in more severe cases, difficulty breathing. You can protect yourself by washing your hands frequently, avoiding touching your face, and avoiding close contact (1 meter or 3 feet) with people who are unwell. An outbreak of COVID-19 started in December 2019 and at the time of the creation of this project was continuing to spread throughout the world. Many governments recommended only essential outings to public places and closed most business that do not serve food or sell essential items. An excellent spatial dashboard built by Johns Hopkins shows the daily confirmed cases by country.

This case study was designed to drive home the important role that data science plays in real-world situations like this pandemic. This case study uses the Random Forest Classifier and a dataset from the South Korean cases of COVID-19 provided on Kaggle to encourage research on this important topic. The goal of the case study is to build a Random Forest Classifier to predict the `state' of the patient.

First, please load the needed packages and modules into Python. Next, load the data into a pandas dataframe for ease of use.

[310]:
```
import os
import pandas as pd
from datetime import datetime,timedelta
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from datetime import timedelta
%matplotlib inline
import plotly.graph_objects as go
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import ExtraTreesRegressor
```

[311]:
```
url ='/Users/josevans/Downloads/RandomForest Covid Case Study_06302020/
 ↪SouthKoreacoronavirusdataset-20200630T044816Z-001.zip'
df = pd.read_csv(url)
df.head()
```

[311]:

	patient_id	global_num	sex	birth_year	age	country	province	city	disease	infection_case	infection_order	infected_by	contact_number	symptom_onset_date	confirmed_date	released_date
0	1000000001	2.0	male	1964.0	50s	Korea	Seoul	Gangseo-gu	NaN	overseas inflow	1.0	NaN	75.0	2020-01-22	2020-01-23	2020-02-05
1	1000000002	5.0	male	1987.0	30s	Korea	Seoul	Jungnang-gu	NaN	overseas inflow	1.0	NaN	31.0	NaN	2020-01-30	2020-03-02
2	1000000003	6.0	male	1964.0	50s	Korea	Seoul	Jongno-gu	NaN	contact with patient	2.0	2.002000e+09	17.0	NaN	2020-01-30	2020-02-19
3	1000000004	7.0	male	1991.0	20s	Korea	Seoul	Mapo-gu	NaN	overseas inflow	1.0	NaN	9.0	2020-01-26	2020-01-30	2020-02-15
4	1000000005	9.0	female	1992.0	20s	Korea	Seoul	Seongbuk-gu	NaN	contact with patient	2.0	1.000000e+09	2.0	NaN	2020-01-31	2020-02-24

```
  deceased_date     state
0           NaN  released
1           NaN  released
2           NaN  released
3           NaN  released
4           NaN  released
```

[312]:
```
df.shape
```

[312]:
```
(2218, 18)
```

[313]:
```
#Counts of null values
na_df=pd.DataFrame(df.isnull().sum().sort_values(ascending=False)).reset_index()
na_df.columns = ['VarName', 'NullCount']
na_df[(na_df['NullCount']>0)]
```

[313]:
```
               VarName  NullCount
0              disease       2199
1        deceased_date       2186
2      infection_order       2176
3   symptom_onset_date       2025
4        released_date       1995
5       contact_number       1807
6          infected_by       1749
7       infection_case       1055
8           global_num        904
9           birth_year        454
10                 age        261
11                 sex        145
12      confirmed_date        141
13               state         88
14                city         65
```

[314]:
```
#counts of response variable values
df.state.value_counts()
```

[314]:
```
isolated    1791
released     307
deceased      32
Name: state, dtype: int64
```

Create a new column named `n_age' which is the calculated age based on the birth year column.

[315]:
```
df['n_age'] = 2020-df.birth_year
```

0.2.1 Handle Missing Values

Print the number of missing values by column.

[316]:
```
df.isnull().sum().sort_values(ascending=False)
```

[316]:
```
disease              2199
deceased_date        2186
infection_order      2176
symptom_onset_date   2025
released_date        1995
contact_number       1807
infected_by          1749
infection_case       1055
global_num            904
birth_year            454
n_age                 454
age                   261
sex                   145
confirmed_date        141
state                  88
city                   65
province                0
country                 0
patient_id              0
dtype: int64
```

[317]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2218 entries, 0 to 2217
Data columns (total 19 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   patient_id          2218 non-null   int64
 1   global_num          1314 non-null   float64
 2   sex                 2073 non-null   object
 3   birth_year          1764 non-null   float64
 4   age                 1957 non-null   object
 5   country             2218 non-null   object
 6   province            2218 non-null   object
 7   city                2153 non-null   object
 8   disease             19 non-null     object
 9   infection_case      1163 non-null   object
 10  infection_order     42 non-null     float64
 11  infected_by         469 non-null    float64
 12  contact_number      411 non-null    float64
 13  symptom_onset_date  193 non-null    object
```

```
 14  confirmed_date      2077 non-null   object
 15  released_date       223 non-null    object
 16  deceased_date       32 non-null     object
 17  state               2130 non-null   object
 18  n_age               1764 non-null   float64
dtypes: float64(6), int64(1), object(12)
memory usage: 329.4+ KB
```

Fill the `disease' missing values with 0 and remap the True values to 1.

[318]:
```
df.disease.fillna(0, inplace=True)
df.disease.replace(True, 1, inplace=True)
```

Fill null values in the following columns with their mean:
`global_number',`birth_year',`infection_order',`infected_by'and 'contact_number'

[319]:
```
cols = ['global_num', 'birth_year', 'infection_order', 'infected_by',␣
 ↪'contact_number']

for col in cols:
    df[col].fillna(df[col].mean(), inplace=True)
```

Fill the rest of the missing values with any method.

[320]:
```
df.isnull().sum().sort_values(ascending=False)
```

[320]:
```
deceased_date         2186
symptom_onset_date    2025
released_date         1995
infection_case        1055
n_age                  454
age                    261
sex                    145
confirmed_date         141
state                   88
city                    65
contact_number           0
infected_by              0
infection_order          0
disease                  0
province                 0
country                  0
birth_year               0
global_num               0
patient_id               0
dtype: int64
```

Check for any remaining null values.

[321]:
```
cols = ['n_age', 'age']

for col in cols:
    df[col].fillna(df.n_age.mean(), inplace=True)

cols = ['infection_case', 'sex', 'state', 'city']

for col in cols:
    df[col].fillna(df[col].value_counts().idxmax(), inplace=True)
```

[322]:
```
df.symptom_onset_date = pd.to_datetime(df.symptom_onset_date)
df.symptom_onset_date.fillna(df.symptom_onset_date.mean(), inplace=True)
df['time_to_recover'] = pd.to_datetime(df.released_date) - pd.to_datetime(df.
 ↪confirmed_date)
time_delta = df.time_to_recover.mean()
df.loc[(df.state != 'deceased'), 'released_date'] = df.loc[(df.state !=␣
 ↪'deceased'), 'released_date'].fillna(pd.to_datetime(df.
 ↪symptom_onset_date)+time_delta)
df.loc[(df.state == 'deceased'), 'released_date'] = 0
df['time_to_death'] = pd.to_datetime(df.deceased_date) - pd.to_datetime(df.
 ↪confirmed_date)
time_delta_death = df.time_to_death.mean()
df.confirmed_date.fillna(df.symptom_onset_date, inplace=True)
df.loc[(df.state != 'deceased'), 'deceased_date'] = 0
df.loc[(df.state == 'deceased'), 'deceased_date'] = df.loc[(df.state ==␣
 ↪'deceased'), 'deceased_date'].fillna(pd.to_datetime(df.
 ↪symptom_onset_date)+time_delta_death)
```

[323]:
```
df.drop(['time_to_recover', 'time_to_death'], axis=1, inplace=True)
```

[324]:
```
df.isnull().sum().sort_values(ascending=False)
```

[324]:
```
n_age                0
disease              0
global_num           0
sex                  0
birth_year           0
age                  0
country              0
province             0
city                 0
infection_case       0
state                0
infection_order      0
infected_by          0
contact_number       0
symptom_onset_date   0
```

```
confirmed_date        0
released_date         0
deceased_date         0
patient_id            0
dtype: int64
```

[325]:
```
df.head(50)
```

[325]:

	patient_id	global_num	sex	birth_year	age	country	province
0	1000000001	2.0	male	1964.0	50s	Korea	Seoul
1	1000000002	5.0	male	1987.0	30s	Korea	Seoul
2	1000000003	6.0	male	1964.0	50s	Korea	Seoul
3	1000000004	7.0	male	1991.0	20s	Korea	Seoul
4	1000000005	9.0	female	1992.0	20s	Korea	Seoul
5	1000000006	10.0	female	1966.0	50s	Korea	Seoul
6	1000000007	11.0	male	1995.0	20s	Korea	Seoul
7	1000000008	13.0	male	1992.0	20s	Korea	Seoul
8	1000000009	19.0	male	1983.0	30s	Korea	Seoul
9	1000000010	21.0	female	1960.0	60s	Korea	Seoul
10	1000000011	23.0	female	1962.0	50s	China	Seoul
11	1000000012	24.0	male	1992.0	20s	Korea	Seoul
12	1000000013	29.0	male	1938.0	80s	Korea	Seoul
13	1000000014	30.0	female	1952.0	60s	Korea	Seoul
14	1000000015	40.0	male	1943.0	70s	Korea	Seoul
15	1000000016	56.0	male	1945.0	70s	Korea	Seoul
16	1000000017	83.0	male	1944.0	70s	Korea	Seoul
17	1000000018	111.0	male	2000.0	20s	Korea	Seoul
18	1000000019	112.0	female	1941.0	70s	Korea	Seoul
19	1000000020	121.0	female	1944.0	70s	Korea	Seoul
20	1000000021	136.0	male	1936.0	80s	Korea	Seoul
21	1000000022	161.0	male	1985.0	30s	Korea	Seoul
22	1000000023	188.0	male	1961.0	50s	Korea	Seoul
23	1000000024	348.0	male	1980.0	40s	Korea	Seoul
24	1000000025	365.0	male	1958.0	60s	Korea	Seoul
25	1000000026	420.0	male	1986.0	30s	Korea	Seoul
26	1000000027	593.0	male	1968.0	50s	Korea	Seoul
27	1000000028	627.0	female	1950.0	70s	Korea	Seoul
28	1000000029	754.0	female	1995.0	20s	Korea	Seoul
29	1000000030	755.0	male	1954.0	60s	China	Seoul
30	1000000031	780.0	male	1965.0	50s	Korea	Seoul
31	1000000032	787.0	male	1962.0	50s	Korea	Seoul
32	1000000033	794.0	female	1970.0	50s	Korea	Seoul
33	1000000034	797.0	male	2000.0	20s	Korea	Seoul
34	1000000035	847.0	male	1984.0	30s	Korea	Seoul
35	1000000036	870.0	female	1963.0	50s	Korea	Seoul
36	1000000037	887.0	female	1976.0	40s	Korea	Seoul
37	1000000038	907.0	male	1953.0	60s	Korea	Seoul

```
38  1000000039       924.0  female      1945.0  70s   China    Seoul
39  1000000040       935.0  female      1960.0  60s   Korea    Seoul
40  1000000041       938.0    male      1968.0  50s   Korea    Seoul
41  1000000042       996.0    male      1977.0  40s   Korea    Seoul
42  1000000043      1022.0  female      1995.0  20s   Korea    Seoul
43  1000000044      1027.0    male      1958.0  60s   Korea    Seoul
44  1000000045      1118.0    male      1979.0  40s   Korea    Seoul
45  1000000046      1246.0  female      1999.0  20s   Korea    Seoul
46  1000000047      1247.0    male      1993.0  20s   Korea    Seoul
47  1000000048      1253.0  female      1995.0  20s   Korea    Seoul
48  1000000049      1254.0    male      1956.0  60s   Korea    Seoul
49  1000000050      1295.0    male      1994.0  20s   Korea    Seoul

            city  disease                infection_case  infection_order  \
0     Gangseo-gu        0              overseas inflow         1.000000
1    Jungnang-gu        0              overseas inflow         1.000000
2      Jongno-gu        0         contact with patient         2.000000
3        Mapo-gu        0              overseas inflow         1.000000
4    Seongbuk-gu        0         contact with patient         2.000000
5      Jongno-gu        0         contact with patient         3.000000
6      Jongno-gu        0         contact with patient         3.000000
7            etc        0              overseas inflow         1.000000
8      Songpa-gu        0              overseas inflow         2.000000
9    Seongbuk-gu        0         contact with patient         3.000000
10  Seodaemun-gu        0              overseas inflow         1.000000
11           etc        0              overseas inflow         1.000000
12     Jongno-gu        0         contact with patient         4.000000
13     Jongno-gu        0         contact with patient         5.000000
14  Seongdong-gu        0             Seongdong-gu APT         2.285714
15     Jongno-gu        0         contact with patient         4.000000
16     Jongno-gu        0         contact with patient         3.000000
17           etc        0                         etc         2.285714
18     Jongno-gu        0         contact with patient         5.000000
19  Seongdong-gu        0             Seongdong-gu APT         2.285714
20     Jongno-gu        0         contact with patient         5.000000
21  Seodaemun-gu        0  Eunpyeong St. Mary's Hospital         2.285714
22     Seocho-gu        0             Shincheonji Church         2.285714
23       Guro-gu        0         contact with patient         2.285714
24   Gangdong-gu        0  Eunpyeong St. Mary's Hospital         2.285714
25     Seocho-gu        0                         etc         2.285714
26    Gangseo-gu        0              overseas inflow         2.285714
27     Jongno-gu        0  Eunpyeong St. Mary's Hospital         2.285714
28     Jongno-gu        0  Eunpyeong St. Mary's Hospital         2.285714
29   Gangdong-gu        0  Eunpyeong St. Mary's Hospital         2.285714
30     Songpa-gu        0                         etc         2.285714
31     Seocho-gu        0         contact with patient         2.285714
32     Songpa-gu        0         contact with patient         2.285714
```

33	Songpa-gu	0	contact with patient	2.285714
34	etc	0	etc	2.285714
35	Eunpyeong-gu	0	Eunpyeong St. Mary's Hospital	2.285714
36	Songpa-gu	0	contact with patient	2.285714
37	etc	0	etc	2.285714
38	Geumcheon-gu	0	overseas inflow	2.285714
39	Gwanak-gu	0	overseas inflow	2.285714
40	Gangdong-gu	0	etc	2.285714
41	Nowon-gu	0	etc	2.285714
42	Songpa-gu	0	overseas inflow	2.285714
43	Dongjak-gu	0	contact with patient	2.285714
44	Songpa-gu	0	contact with patient	2.285714
45	etc	0	contact with patient	2.285714
46	Gangnam-gu	0	Shincheonji Church	2.285714
47	Eunpyeong-gu	0	etc	2.285714
48	Eunpyeong-gu	0	Eunpyeong St. Mary's Hospital	2.285714
49	Gwanak-gu	0	etc	2.285714

	infected_by	contact_number	symptom_onset_date	confirmed_date	\
0	2.600789e+09	75.000000	2020-01-22 00:00:00.000000000	2020-01-23	
1	2.600789e+09	31.000000	2020-02-29 05:43:12.746114048	2020-01-30	
2	2.002000e+09	17.000000	2020-02-29 05:43:12.746114048	2020-01-30	
3	2.600789e+09	9.000000	2020-01-26 00:00:00.000000000	2020-01-30	
4	1.000000e+09	2.000000	2020-02-29 05:43:12.746114048	2020-01-31	
5	1.000000e+09	43.000000	2020-02-29 05:43:12.746114048	2020-01-31	
6	1.000000e+09	0.000000	2020-02-29 05:43:12.746114048	2020-01-31	
7	2.600789e+09	0.000000	2020-02-29 05:43:12.746114048	2020-02-02	
8	2.600789e+09	68.000000	2020-02-29 05:43:12.746114048	2020-02-05	
9	1.000000e+09	6.000000	2020-02-29 05:43:12.746114048	2020-02-05	
10	2.600789e+09	23.000000	2020-02-29 05:43:12.746114048	2020-02-06	
11	2.600789e+09	0.000000	2020-02-29 05:43:12.746114048	2020-02-07	
12	1.000000e+09	117.000000	2020-02-29 05:43:12.746114048	2020-02-16	
13	1.000000e+09	27.000000	2020-02-06 00:00:00.000000000	2020-02-16	
14	2.600789e+09	8.000000	2020-02-11 00:00:00.000000000	2020-02-19	
15	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-19	
16	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-20	
17	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-20	
18	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-20	
19	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-20	
20	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-20	
21	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-21	
22	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-21	
23	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-22	
24	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-22	
25	2.600789e+09	24.128954	2020-02-21 00:00:00.000000000	2020-02-22	
26	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-23	
27	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-23	

28	1.000000e+09	24.128954	2020-02-11 00:00:00.000000000	2020-02-26
29	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-23
30	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-22
31	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-23
32	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-24
33	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-24
34	2.600789e+09	24.128954	2020-02-19 00:00:00.000000000	2020-02-25
35	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-25
36	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-25
37	2.600789e+09	24.128954	2020-02-24 00:00:00.000000000	2020-02-25
38	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-25
39	1.500000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-25
40	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-25
41	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
42	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
43	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
44	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
45	1.000000e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
46	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
47	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
48	2.600789e+09	24.128954	2020-02-29 05:43:12.746114048	2020-02-26
49	2.600789e+09	24.128954	2020-02-24 00:00:00.000000000	2020-02-27

	released_date	deceased_date	state	n_age
0	2020-02-05	0	released	56.0
1	2020-03-02	0	released	33.0
2	2020-02-19	0	released	56.0
3	2020-02-15	0	released	29.0
4	2020-02-24	0	released	28.0
5	2020-02-19	0	released	54.0
6	2020-02-10	0	released	25.0
7	2020-02-24	0	released	28.0
8	2020-02-21	0	released	37.0
9	2020-02-29	0	released	60.0
10	2020-02-29	0	released	58.0
11	2020-02-27	0	released	28.0
12	2020-03-14 01:37:49.876158891	0	released	82.0
13	2020-03-12	0	released	68.0
14	2020-02-24 19:54:37.130044843	0	isolated	77.0
15	2020-03-11	0	released	75.0
16	2020-03-01	0	released	76.0
17	2020-03-14 01:37:49.876158891	0	isolated	20.0
18	2020-03-08	0	released	79.0
19	2020-03-14 01:37:49.876158891	0	isolated	76.0
20	2020-03-08	0	released	84.0
21	2020-03-14 01:37:49.876158891	0	isolated	35.0
22	2020-03-14 01:37:49.876158891	0	isolated	59.0

```
23                    2020-03-14            0  released   40.0
24  2020-03-14 01:37:49.876158891            0  isolated   62.0
25                    2020-03-11            0  released   34.0
26                    2020-03-04            0  released   52.0
27                    2020-03-11            0  released   70.0
28                    2020-03-11            0  released   25.0
29  2020-03-14 01:37:49.876158891            0  released   66.0
30                    2020-03-19            0  released   55.0
31  2020-03-14 01:37:49.876158891            0  isolated   58.0
32                    2020-03-10            0  released   50.0
33                    2020-03-17            0  released   20.0
34  2020-03-03 19:54:37.130044843            0  isolated   36.0
35  2020-03-14 01:37:49.876158891            0  isolated   57.0
36                    2020-03-11            0  released   44.0
37  2020-03-08 19:54:37.130044843            0  isolated   67.0
38  2020-03-14 01:37:49.876158891            0  released   75.0
39  2020-03-14 01:37:49.876158891            0  released   60.0
40                    2020-03-04            0  released   52.0
41  2020-03-14 01:37:49.876158891            0  isolated   43.0
42  2020-03-14 01:37:49.876158891            0  released   25.0
43  2020-03-14 01:37:49.876158891            0  isolated   62.0
44                    2020-03-04            0  released   41.0
45                    2020-03-03            0  released   21.0
46                    2020-03-11            0  released   27.0
47                    2020-03-13            0  released   25.0
48  2020-03-14 01:37:49.876158891            0  isolated   64.0
49  2020-03-08 19:54:37.130044843            0  isolated   26.0
```

Remove date columns from the data.

[326]:
```
df = df.
↪drop(['symptom_onset_date','confirmed_date','released_date','deceased_date'],axis␣
↪=1)
```

Review the count of unique values by column.

[297]:
```
print(df.nunique())
```

```
patient_id         2218
global_num         1304
sex                   2
birth_year           97
age                  12
country               4
province             17
city                134
disease               2
infection_case       16
```

```
infection_order       7
infected_by         207
contact_number       73
state                 3
n_age                97
dtype: int64
```

Review the percent of unique values by column.

[329]:
```
print(df.nunique()/df.shape[0])
```

```
patient_id          1.000000
global_num          0.587917
sex                 0.000902
birth_year          0.043733
age                 0.005410
country             0.001803
province            0.007665
city                0.060415
disease             0.000902
infection_case      0.007214
infection_order     0.003156
infected_by         0.093327
contact_number      0.032913
state               0.001353
n_age               0.043733
dtype: float64
```

Review the range of values per column.

[328]:
```
df.describe().T
```

[328]:

	count	mean	std	min	\
patient_id	2218.0	4.014678e+09	2.192419e+09	1.000000e+09	
global_num	2218.0	4.664817e+03	2.211785e+03	1.000000e+00	
birth_year	2218.0	1.974989e+03	1.731123e+01	1.916000e+03	
disease	2218.0	8.566276e-03	9.217769e-02	0.000000e+00	
infection_order	2218.0	2.285714e+00	1.706622e-01	1.000000e+00	
infected_by	2218.0	2.600789e+09	7.216328e+08	1.000000e+09	
contact_number	2218.0	2.412895e+01	3.917141e+01	0.000000e+00	
n_age	2218.0	4.501134e+01	1.731123e+01	0.000000e+00	

	25%	50%	75%	max
patient_id	1.700000e+09	6.001000e+09	6.004000e+09	7.000000e+09
global_num	4.205250e+03	4.664817e+03	5.900250e+03	8.717000e+03
birth_year	1.965000e+03	1.974989e+03	1.988000e+03	2.020000e+03
disease	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
infection_order	2.285714e+00	2.285714e+00	2.285714e+00	6.000000e+00
infected_by	2.600789e+09	2.600789e+09	2.600789e+09	6.113000e+09

```
contact_number   2.412895e+01  2.412895e+01  2.412895e+01  1.160000e+03
n_age            3.200000e+01  4.501134e+01  5.500000e+01  1.040000e+02
```

0.2.2 Check for duplicated rows

[327]:
```
duplicateRowsDF = df[df.duplicated()]
duplicateRowsDF
```

[327]:
```
Empty DataFrame
Columns: [patient_id, global_num, sex, birth_year, age, country, province, city,
disease, infection_case, infection_order, infected_by, contact_number, state,
n_age]
Index: []
```

Print the categorical columns and their associated levels.

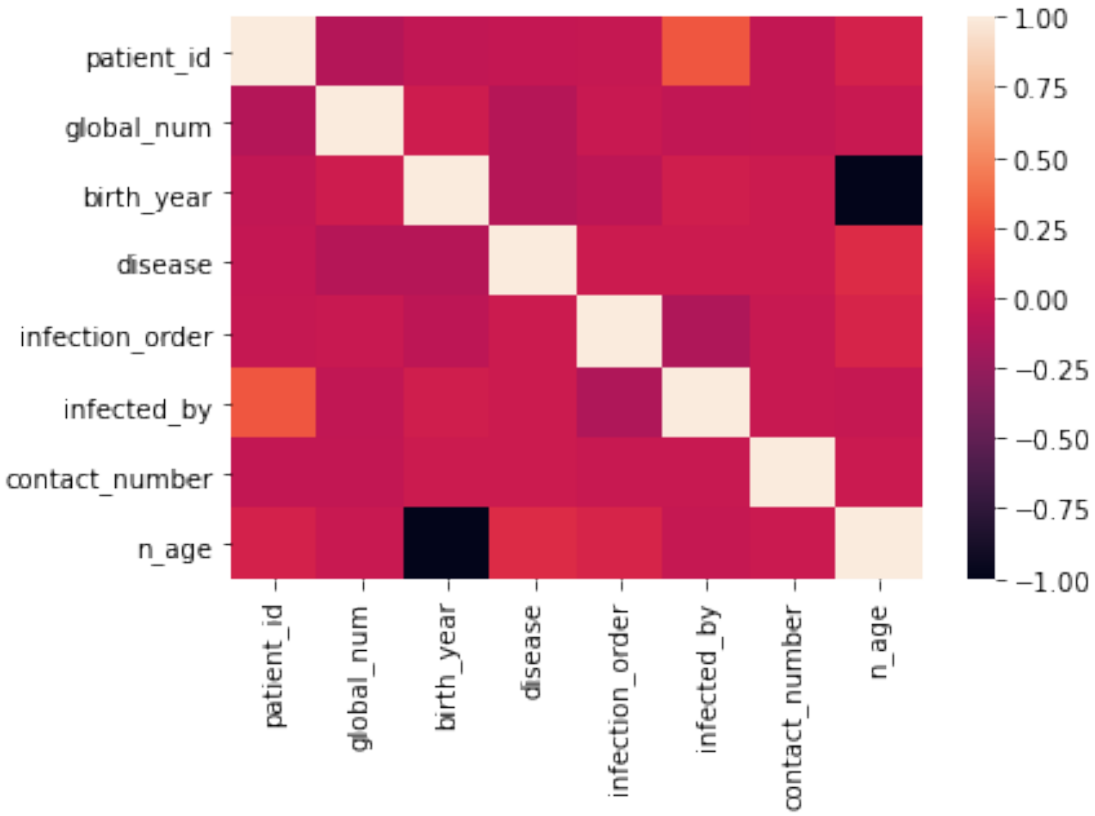
[301]:
```
dfo = df.select_dtypes(include=['object'], exclude=['datetime'])
dfo.shape
#get levels for all variables
vn = pd.DataFrame(dfo.nunique()).reset_index()
vn.columns = ['VarName', 'LevelsCount']
vn.sort_values(by='LevelsCount', ascending =False)
vn
```

[301]:

	VarName	LevelsCount
0	sex	2
1	age	12
2	country	4
3	province	17
4	city	134
5	infection_case	16
6	state	3

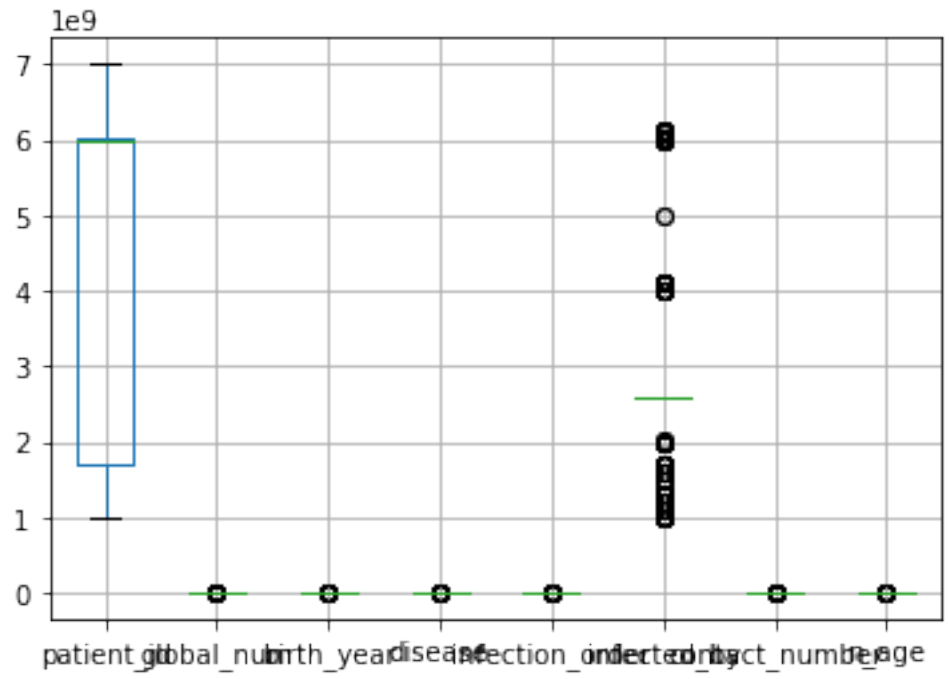
Plot the correlation heat map for the features.

[330]:
```
ax = sns.heatmap(df.corr())
```



Plot the boxplots to check for outliers.

[304]:
```
df.boxplot()
plt.show()
```



Create dummy features for object type features.

[333]:
```
df_dummies = pd.get_dummies(df)
df_dummies
```

[333]:

	patient_id	global_num	birth_year	disease	infection_order	\
0	1000000001	2.000000	1964.0	0	1.000000	
1	1000000002	5.000000	1987.0	0	1.000000	
2	1000000003	6.000000	1964.0	0	2.000000	
3	1000000004	7.000000	1991.0	0	1.000000	
4	1000000005	9.000000	1992.0	0	2.000000	
…	…	…	…	…	…	
2213	6100000085	4664.816591	1990.0	0	2.285714	
2214	7000000001	139.000000	1998.0	0	2.285714	
2215	7000000002	222.000000	1998.0	0	2.285714	
2216	7000000003	4345.000000	1972.0	0	2.285714	
2217	7000000004	5534.000000	1974.0	0	2.285714	

	infected_by	contact_number	n_age	sex_female	sex_male	…	\
0	2.600789e+09	75.000000	56.0	0	1	…	
1	2.600789e+09	31.000000	33.0	0	1	…	
2	2.002000e+09	17.000000	56.0	0	1	…	
3	2.600789e+09	9.000000	29.0	0	1	…	
4	1.000000e+09	2.000000	28.0	1	0	…	
…	…	…	…	…	…	…	

```
2213  2.600789e+09       24.128954   30.0           0         1  …
2214  2.600789e+09       87.000000   22.0           0         1  …
2215  2.600789e+09       84.000000   22.0           1         0  …
2216  2.600789e+09       21.000000   48.0           1         0  …
2217  2.600789e+09       74.000000   46.0           0         1  …

      infection_case_Shincheonji Church  \
0                                     0
1                                     0
2                                     0
3                                     0
4                                     0
…                                   …
2213                                  0
2214                                  0
2215                                  0
2216                                  0
2217                                  0

      infection_case_Suyeong-gu Kindergarten  \
0                                          0
1                                          0
2                                          0
3                                          0
4                                          0
…                                        …
2213                                       0
2214                                       0
2215                                       0
2216                                       0
2217                                       0

      infection_case_contact with patient  infection_case_etc  \
0                                       0                   0
1                                       0                   0
2                                       1                   0
3                                       0                   0
4                                       1                   0
…                                     …                   …
2213                                    1                   0
2214                                    0                   1
2215                                    0                   1
2216                                    0                   1
2217                                    0                   1

      infection_case_gym facility in Cheonan  \
0                                          0
```

```
1                                          0
2                                          0
3                                          0
4                                          0
…                                        …
2213                                       0
2214                                       0
2215                                       0
2216                                       0
2217                                       0

      infection_case_gym facility in Sejong  infection_case_overseas inflow  \
0                                          0                                1
1                                          0                                1
2                                          0                                0
3                                          0                                1
4                                          0                                0
…                                        …                                …
2213                                       0                                0
2214                                       0                                0
2215                                       0                                0
2216                                       0                                0
2217                                       0                                0

      state_deceased  state_isolated  state_released
0                  0               0               1
1                  0               0               1
2                  0               0               1
3                  0               0               1
4                  0               0               1
…                …               …               …
2213               0               1               0
2214               0               1               0
2215               0               0               1
2216               0               0               1
2217               0               1               0

[2218 rows x 196 columns]
```

0.2.3 Split the data into test and train subsamples

[338]:
```
from sklearn.model_selection import train_test_split

# dont forget to define your X and y

X = df_dummies.drop(['state_deceased', 'state_isolated', 'state_released'],␣
 ↪axis=1)
```

```
y = df.state

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2,␣
 ↪random_state=1)
```

0.2.4 Scale data to prep for model creation

[339]:
```
#scale data
from sklearn import preprocessing
import numpy as np
# build scaler based on training data and apply it to test data to then also␣
 ↪scale the test data
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

[340]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import␣
 ↪classification_report,confusion_matrix,roc_curve,roc_auc_score
from sklearn.metrics import accuracy_score,log_loss
from matplotlib import pyplot
```

0.2.5 Fit Random Forest Classifier

The fit model shows an overall accuracy of 80% which is great and indicates our model was effectively able to identify the status of a patients in the South Korea dataset.

[341]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=300, random_state = 1,n_jobs=-1)
model_res = clf.fit(X_train_scaled, y_train)
y_pred = model_res.predict(X_test_scaled)
y_pred_prob = model_res.predict_proba(X_test_scaled)
lr_probs = y_pred_prob[:,1]
ac = accuracy_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)
```

```
print('Random Forest: Accuracy=%.3f' % (ac))

print('Random Forest: f1-score=%.3f' % (f1))
```

```
Random Forest: Accuracy=0.863
Random Forest: f1-score=0.831
```

0.2.6 Create Confusion Matrix Plots

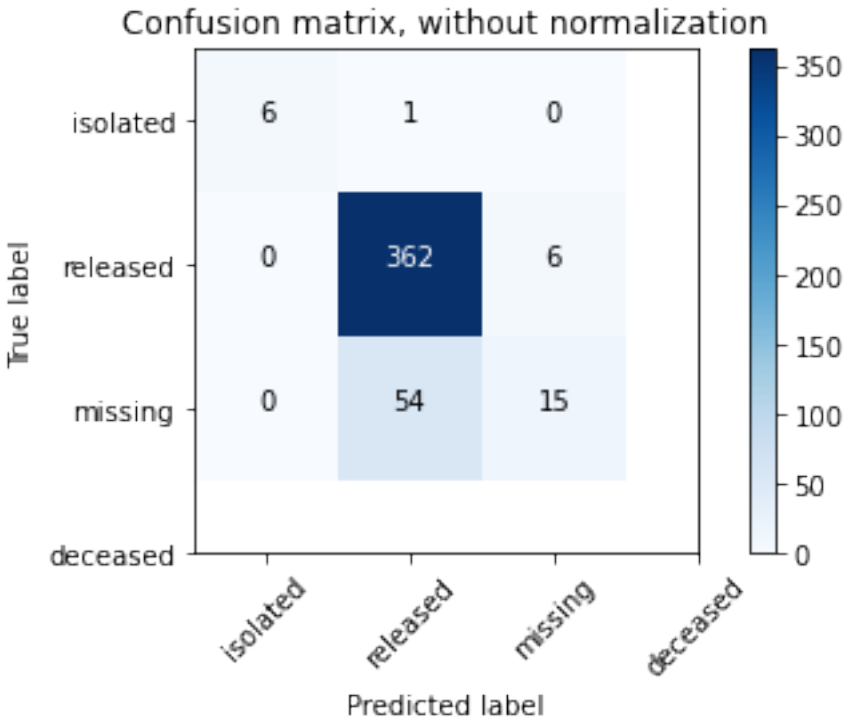
Confusion matrices are great ways to review your model performance for a multi-class classification problem. Being able to identify which class the misclassified observations end up in is a great way to determine if you need to build additional features to improve your overall model. In the example below we plot a regular counts confusion matrix as well as a weighted percent confusion matrix. The percent confusion matrix is particulary helpful when you have unbalanced class sizes.

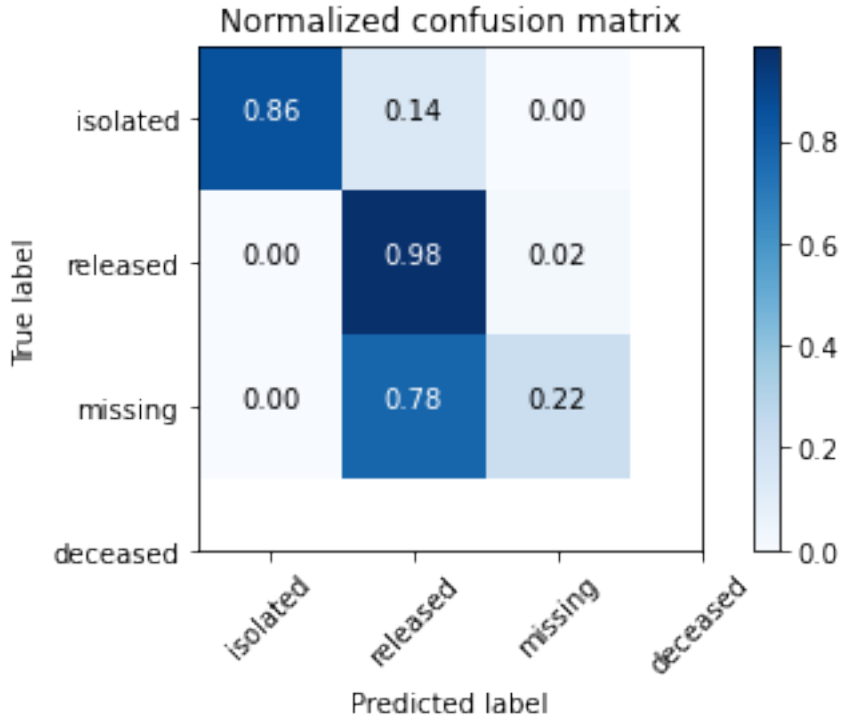
[342]:
```
class_names=['isolated','released','missing','deceased'] # name  of classes
```

[343]:
```
import itertools
import numpy as np
import matplotlib.pyplot as plt

from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
```

```
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')
#plt.savefig('figures/RF_cm_multi_class.png')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')
#plt.savefig('figures/RF_cm_proportion_multi_class.png', bbox_inches="tight")
plt.show()
```

```
Confusion matrix, without normalization
[[  6   1   0]
 [  0 362   6]
 [  0  54  15]]
Normalized confusion matrix
[[0.86 0.14 0.  ]
 [0.   0.98 0.02]
 [0.   0.78 0.22]]
```


Confusion matrix, without normalization
isolated
released
missing
deceased
True label
6
1
0
0
362
6
0
54
15
350
300
250
200
150
100
50
0
isolated
released
missing
deceased
Predicted label

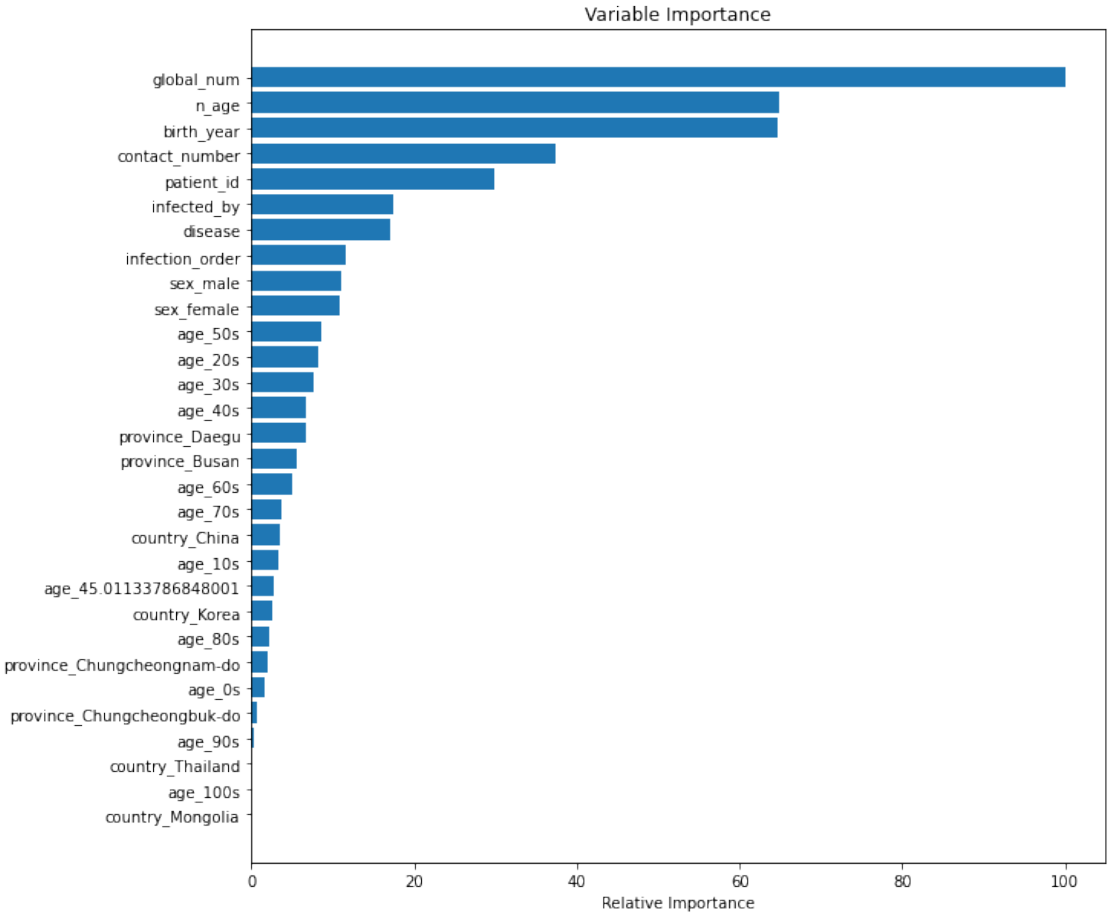

Normalized confusion matrix
isolated
released
missing
deceased
True label
0.86
0.14
0.00
0.00
0.98
0.02
0.00
0.78
0.22
0.8
0.6
0.4
0.2
0.0
isolated
released
missing
deceased
Predicted label

0.2.7 Plot feature importances

The random forest algorithm can be used as a regression or classification model. In either case it tends to be a bit of a black box, where understanding what's happening under the hood can be difficult. Plotting the feature importances is one way that you can gain a perspective on which features are driving the model predictions.

[344]:
```
feature_importance = clf.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.max())[:
 ↪30]
sorted_idx = np.argsort(feature_importance)[:30]

pos = np.arange(sorted_idx.shape[0]) + .5
print(pos.size)
sorted_idx.size
plt.figure(figsize=(10,10))
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, X.columns[sorted_idx])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()
```

```
30
```



The popularity of random forest is primarily due to how well it performs in a multitude of data situations. It tends to handle highly correlated features well, where as a linear regression model would not. In this case study we demonstrate the performance ability even with only a few features and almost all of them being highly correlated with each other. Random Forest is also used as an efficient way to investigate the importance of a set of features with a large data set. Consider random forest to be one of your first choices when building a decision tree, especially for multiclass classifications.