Estimating COVID-19's R_t in Real-Time

Kevin Systrom - Kalman Rupp - Mike Krieger

Cross-posted, adapted from Kevin Systrom's adapted version of Kevin Systrom's notebook that shows his excellent R-hat update.

Docker: mathematicalmonk/jupyter-ds-kevin:3-cmdstan Kevin's Docker is Cmdstan v2 but Mac's Homebrew is now v3, so this uses the latest Cmdstan v3.

Regular updates with the latest changes can be found at https://github.com/mathematicalmonk/covid19-analysis/commits/master

Discuss updates with the latest changes on the forum at https://forum.mathematicalmonk.com/

Estimating COVID-19's R_t in Real-Time

Kevin Systrom - Rt

Cross-posted, adapted from Mike Krieger's adapted version of Kevin Systrom's notebook that shows his excellent R-hat update.

Docker: mathematicalmonk/jupyter-ds-kevin:3-cmdstan Kevin's Docker is Cmdstan v2 but Mac's Homebrew is now v3, so this uses the latest Cmdstan v3.

Regular updates with the latest changes can be found at https://github.com/mathematicalmonk/covid19-analysis/commits/master

Discuss updates with the latest changes on the forum at https://forum.mathematicalmonk.com/covid19-analysis/rt-estimate-covid-19

Kevin Systrom - Rt

Cross-posted, adapted from Mike Krieger's adapted version of Kevin Systrom's notebook that shows his excellent R-hat update.

Docker: mathematicalmonk/jupyter-ds-kevin:3-cmdstan Kevin's Docker is Cmdstan v2 but Mac's Homebrew is now v3, so this uses the latest Cmdstan v3.

Regular updates with the latest changes can be found at https://github.com/mathematicalmonk/covid19-analysis/commits/master

Discuss updates with the latest changes on the forum at https://forum.mathematicalmonk.com/covid19-analysis/rt-estimate-covid-19

The value of R_t before and in the crisis, t is defined as contact force β Effective on measures have been continuing as a whole and β is given as the information about whether we should increase or reduce restrictions based on our prevailing goals of economic prosperity and human safety. Rich Associate Professor of Statistics at Duke University, Author of Bayesian Data Analysis

So, today, we can get an R_t of this day. In fact, the only data that matters! You can find the Rt.live Rt.live Rt.live, Rt.live, and Rt.live for understanding R_t at a national level. Instead, to change this crisis effectively, we need a local state, county level, and precisely R_t.

Fortunately, Kevin Systrom, co-founder of Instagram, has made great strides in estimating R_t for the United States. Kevin's approach is optimized for speed, and he has made it very easy to run his model. Kevin's work has been instrumental in shaping our own approach.

If you have questions, comments, or improvements feel free to give a hand. Starting on March, and this is the entire state. You can at epidemiologist. At the same time, we are a team, and we are attempting to be of great use to our local epidemiologists so as we collaborate with them, we can use the same language. We are a team of data scientists, and we are not epidemiologists. We are a team of data scientists, and we are not epidemiologists. We are a team of data scientists, and we are not epidemiologists

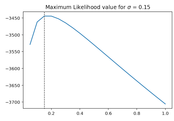
```
        # Holds all posteriors with every given value of sigma
        result['posteriors'] = []

        # Holds the log likelihood across all k for each value of sigma
        result['log_likelihoods'] = []

        for sigma in sigmas:
            posteriors, log_likelihood = get_posteriors(smoothed, sigma=sigma)
            result['posteriors'].append(posteriors)
            result['log_likelihoods'].append(log_likelihood)

        # Store all results keyed off of state name
        results[region_name] = result
        clear_output(wait=True)

    except:
        pass

print('Done.')
```

```
Done.
```

Now that we have all the log likelihoods, we can sum for each value of sigma across states, graph it, then choose the maximum.

In [19]:
```
# Each index of this array holds the total of the log likelihoods for
# the corresponding index of the sigmas array.
total_log_likelihoods = np.zeros_like(sigmas)

# Loop through each state's results and add the log likelihoods to the running total.
for region_name, result in results.items():
    total_log_likelihoods += result['log_likelihoods']

# Select the index with the largest log likelihood total
max_likelihood_index = total_log_likelihoods.argmax()

# Select the value that has the highest log likelihood
sigma = sigmas[max_likelihood_index]

# Plot it
fig, ax = plt.subplots()
ax.set_title(f"Maximum Likelihood value for $\sigma$ = {sigma:.2f}");
ax.plot(sigmas, total_log_likelihoods)
ax.axvline(sigma, color='k', linestyle=':')
```

Out[19]:
```
<matplotlib.lines.Line2D at 0x1113a6710>
```



Compile Final Results

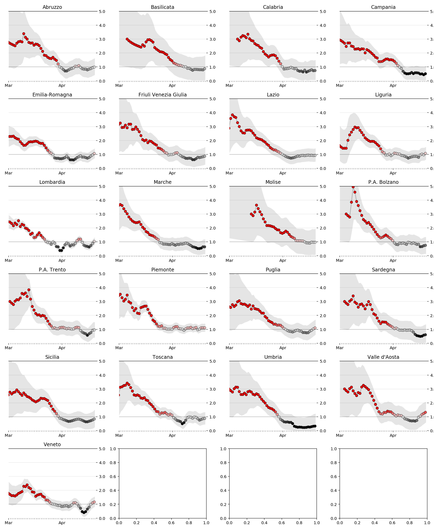
Given that we've selected the *optimal* σ, let's grab the *precalculated* posterior corresponding to that value of σ for each state. Let's also calculate the 90% and 50% highest density intervals (this takes a little while) and also the most likely value.

In [20]:
```
final_results = None

for region_name, result in results.items():
    try:

        print(region_name)
        posteriors = result['posteriors'][max_likelihood_index]
        hdis_90 = highest_density_interval(posteriors, p=.9)
        hdis_50 = highest_density_interval(posteriors, p=.5)
        most_likely = posteriors.idxmax().rename('ML')
        result = pd.concat([most_likely, hdis_90, hdis_50], axis=1)
        if final_results is None:
            final_results = result
        else:
            final_results = pd.concat([final_results, result])
        clear_output(wait=True)

    except:
        pass

print('Done.')
```

```
Done.
```

Plot All regions

In [21]:
```
ncols = 4
nrows = int(np.ceil(len(results) / ncols))

fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(15, nrows*3))

for i, (region_name, result) in enumerate(final_results.groupby('denominazione_regione')):
    try:
        plot_rt(result, axes.flat[i], region_name)
    except:
        pass

fig.tight_layout()
fig.set_facecolor('w')
fig.savefig('/Users/Alastair/Documents/Jupyter/covid-19-italy/Daily Results/'+timestr+'/AllregionsRt_'+ timestr+'.png'
)
```



Export Data to CSV

In [ ]:
```
# Uncommmet the following line if you'd like to export the data
#final_results.to_csv('data/rt.csv')
```

Standings

In [ ]:
```
# As of 4/15. This can be extended with all regions.
# In this case, I only entered some European regions.
no_lockdown = [

]
partial_lockdown = [
#    'Lombardia',


]

FULL_COLOR = [.7,.7,.7]
NONE_COLOR = [179/255,35/255,14/255]
PARTIAL_COLOR = [.5,.5,.5]
ERROR_BAR_COLOR = [.3,.3,.3]
```

filtered = overall.index.get_level_values(0).isin(FILTERED_REGIONS) mr = overall.loc[~filtered].groupby(level=0)[['ML', 'High', 'Low']].last()

def plot_standings(mr, figsize=None, title='Most Recent R_t by Country'): if not figsize: figsize = ((15.9/50)*len(mr)+.1,2.5)

```
    fig, ax = plt.subplots(figsize=figsize)

    ax.set_title(title)
    err = mr[['Low', 'High']].sub(mr['ML'], axis=0).abs()
    bars = ax.bar(mr.index,
                  mr['ML'],
                  width=.825,
                  color=FULL_COLOR,
                  ecolor=ERROR_BAR_COLOR,
                  capsize=2,
                  error_kw={'alpha':.5, 'lw':1},
                  yerr=err.values.T)

    for bar, region_name in zip(bars, mr.index):
        if region_name in no_lockdown:
            bar.set_color(NONE_COLOR)
        if region_name in partial_lockdown:
            bar.set_color(PARTIAL_COLOR)

    labels = mr.index.to_series()
    ax.set_xticklabels(labels, rotation=90, fontsize=11)
    ax.margins(0)
    ax.set_ylim(0,2.)
    ax.axhline(1.0, linestyle=':', color='k', lw=1)

    leg = ax.legend(handles=[
                        Patch(label='Full', color=FULL_COLOR),
                        Patch(label='Partial', color=PARTIAL_COLOR),
                        Patch(label='None', color=NONE_COLOR)
                    ],
                    title='Lockdown',
                    ncol=3,
                    loc='upper left',
                    columnspacing=.75,
                    handletextpad=.5,
                    handlelength=1)

    leg._legend_box.align = "left"
    fig.set_facecolor('w')
    return fig, ax
```

mr.sort_values('ML', inplace=True) plotstandings(mr); plt.savefig('/Users/Alastair/Documents/Jupyter/covid-19-italy/Daily Results/'+timestr+'/MostLikelyUnsorted'+ timestr+'.png', bbox_inches = 'tight', dpi=300)

In [ ]:
```
filtered = final_results.index.get_level_values(0).isin(FILTERED_REGIONS)
mr = final_results.loc[~filtered].groupby(level=0)[['ML', 'High_90', 'Low_90']].last()

def plot_standings(mr, figsize=None, title='Most Recent $R_t$ by Country'):
    if not figsize:
        figsize = ((15.9/50)*len(mr)+.1,2.5)

    fig, ax = plt.subplots(figsize=figsize)

    ax.set_title(title)
    err = mr[['Low_90', 'High_90']].sub(mr['ML'], axis=0).abs()
    bars = ax.bar(mr.index,
                  mr['ML'],
                  width=.825,
                  color=FULL_COLOR,
                  ecolor=ERROR_BAR_COLOR,
                  capsize=2,
                  error_kw={'alpha':.5, 'lw':1},
                  yerr=err.values.T)

    for bar, region_name in zip(bars, mr.index):
        if region_name in no_lockdown:
            bar.set_color(NONE_COLOR)
        if region_name in partial_lockdown:
            bar.set_color(PARTIAL_COLOR)

    labels = mr.index.to_series()
    ax.set_xticklabels(labels, rotation=90, fontsize=11)
    ax.margins(0)
    ax.set_ylim(0,2.)
    ax.axhline(1.0, linestyle=':', color='k', lw=1)

    leg = ax.legend(handles=[
                        Patch(label='Full', color=FULL_COLOR),
                        Patch(label='Partial', color=PARTIAL_COLOR),
                        Patch(label='None', color=NONE_COLOR)
                    ],
                    title='Lockdown',
                    ncol=3,
                    loc='upper left',
                    columnspacing=.75,
                    handletextpad=.5,
                    handlelength=1)

    leg._legend_box.align = "left"
    fig.set_facecolor('w')
    return fig, ax

mr.sort_values('ML', inplace=True)
plot_standings(mr);
```

In [ ]:
```
mr.sort_values('High_90', inplace=True)
plot_standings(mr);
plt.savefig('/Users/Alastair/Documents/Jupyter/covid-19-italy/Daily Results/'+timestr+'/MostLikelySorted_'+ timestr+'.
png', bbox_inches = 'tight', dpi=300)
```

In [ ]:
```
show = mr[mr.High_90.le(1.1)].sort_values('ML')
fig, ax = plot_standings(show, title='Likely Under Control');
fig.savefig('/Users/Alastair/Documents/Jupyter/covid-19-italy/Daily Results/'+timestr+'/LikelyUnderControl_'+ timestr+
'.png', bbox_inches = 'tight', dpi=300)
```

In [ ]:
```
show = mr[mr.Low_90.ge(1.05)].sort_values('Low_90')
fig, ax = plot_standings(show, title='Likely Not Under Control');
ax.get_legend().remove()
fig.savefig('/Users/Alastair/Documents/Jupyter/covid-19-italy/Daily Results/'+timestr+'/LikelyNotUnderControl_'+ times
tr'+'.png', bbox_inches = 'tight', dpi=300)
```

In [ ]:

In [ ]: