Random Forest Fires: Modeling Wildfire Probability with Weather

Matt Elser
elserm@gmail.com

September 8, 2020

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor. Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in. Nunc fermentum nisi imperdiet lectus tincidunt vestibulum at ac elit. Nulla mattis nisl eu malesuada suscipit.

$$I = \int_a^b f(x)\,\mathrm{d}x. \tag{1}$$

1 Requirements

Running the code requires the following modules:

- `pandas`
- `numpy`
- `sklearn`
- `requests` - for accessing FCC area information API
- `plotly` - for creating graphs of geographic areas
- `kaleido` - required for plotly to write graphs to images

As well as the code and data available at https://github.com/mattelser/randomForestFires. The tool `ffmpeg` was used to combine graphs into an animation, viewable at the aforementioned github link. All work was done on a unix system and no validation was feasible on windows.

2 Data Gathering

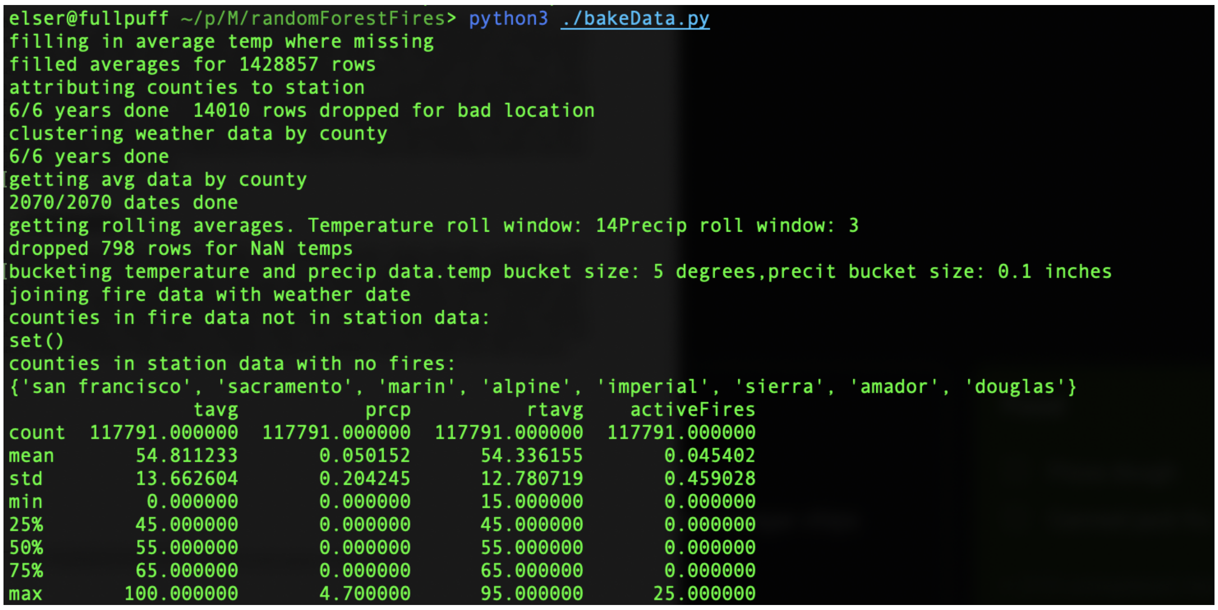
Weather data is readily available from the National Climatic Data Center (NCDC) and the National Oceanic and Atmospheric Administration (NOAA), and wildfire data is made available by the California Department of Forestry and Fire Protection (CalFire), however obtaining sufficient data in a usable form still proved challenging. Raw data from NOAA [1] (available in `/rawData/weather`) had to be obtained through multiple manually-clicked requests, as no API existed to query and a limit on request size meant several requests had to be made per year of data. CalFire makes their data available via "Redbook" pdfs [2] , from which the contents of tables was manually extracted using a tool called `Tabula`. The fire data is only by county, while the weather data was by station, which had no county attributed. Separately, a collection of latitudes and longitudes was obtained (again from NCDC/NOAA) for each station. This allowed for obtaining county via an API made available by the Federal Communications Comission (FCC) [3] , which accepts

[1] https://www.ncdc.noaa.gov/cdo-web/search
[2] https://www.fire.ca.gov/stats-events/
[3] https://geo.fcc.gov/api/census/#!/area/get_area

a latitude/longitude pair and returns geographic information. Included in this information is both county name and Federal Information Processing Standards (FIPS) code, which was one key part in graphing information by county. The second key part to graphing by county is pairing each FIPS code with a polygonal outline of that county (defined by points of latitude/longitude). This was available via an example in the documentation for the graphing module `plotly` [4] in the form of a "geojson" file defining the shape of all counties in the US. A paired-down version of this geojson file containing only California is available in `//CalCountyGeoJson.json`.

3 Code Description

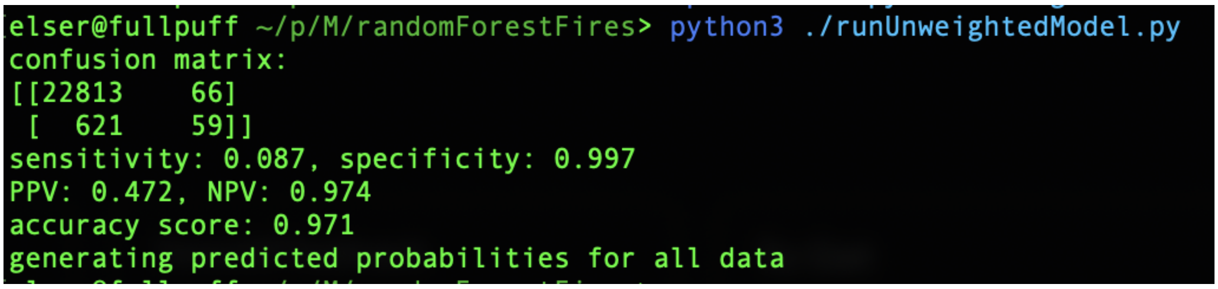
The bulk of the code centers around joining, clustering, cleaning, bucketing, and extrapolating from the data. This includes all data gathering steps mentioned above, other than those explicitly mentioned as being manual (requesting weather data and extracting tables from CalFire pdfs). The data pipeline starts with these manually extracted data files, then develops them in stages, "baking" the results out in between. First, `bakeData.py` takes all the per-station-per-date observations from `\cleanedData` and generates average per-county data for each date, as well as a rolling average of temperature and the recent presence of precipitation per county. Measurements are also bucketed for better usability in a random forest model. Once we have observations per county per date, we join the fire data, which begins as various fields per-fire (notably, county, start date, and date of containment). We end up with a data frame where each row is an observation of one county on a given day, including weather and how many fires are burning that day. A description of each action is printed to stdout:



```
elser@fullpuff ~/p/M/randomForestFires> python3 ./bakeData.py
filling in average temp where missing
filled averages for 1428857 rows
attributing counties to station
6/6 years done  14010 rows dropped for bad location
clustering weather data by county
6/6 years done
getting avg data by county
2070/2070 dates done
getting rolling averages. Temperature roll window: 14Precip roll window: 3
dropped 798 rows for NaN temps
bucketing temperature and precip data.temp bucket size: 5 degrees,precit bucket size: 0.1 inches
joining fire data with weather date
counties in fire data not in station data:
set()
counties in station data with no fires:
{'san francisco', 'sacramento', 'marin', 'alpine', 'imperial', 'sierra', 'amador', 'douglas'}
                tavg           prcp          rtavg    activeFires
count  117791.000000  117791.000000  117791.000000  117791.000000
mean       54.811233       0.050152      54.336155       0.045402
std        13.662604       0.204245      12.780719       0.459028
min         0.000000       0.000000      15.000000       0.000000
25%        45.000000       0.000000      45.000000       0.000000
50%        55.000000       0.000000      55.000000       0.000000
75%        65.000000       0.000000      65.000000       0.000000
max       100.000000       4.700000      95.000000      25.000000
```

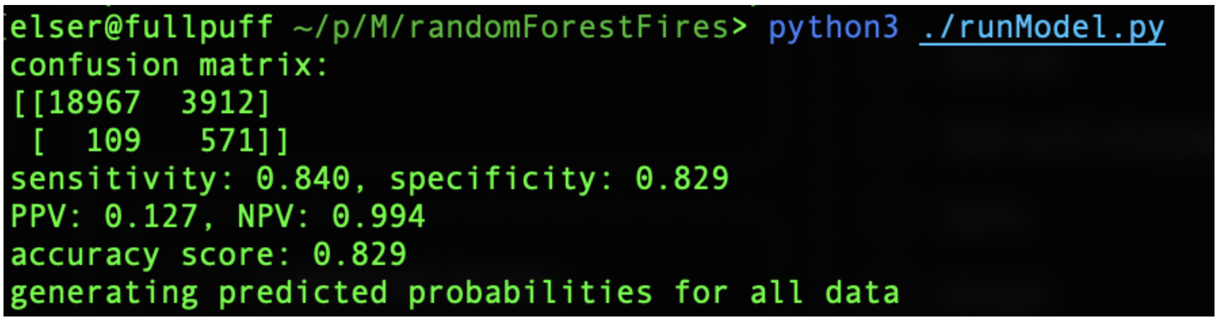
This is the data that feeds the model. A `RandomForestClassifier` is fed per county per day observations of average temperature, precipitation level, a rolling average of temperature, whether there has recently been precipitation, and what county these observations were in. Notably, date is not given to the model, even though these observations are per-date, because dates do not contribute to fires. The target is whether there is a fire in that county. Initial runs of the model were very good at predicting when/where there would not be a fire, and mediocre at predicting when/where a fire would occur. Fires are relatively rare in the dataset, so it makes sense that they would not be predicted often, however in this case false negatives could be catastrophic and therefore reweighting of the model was necessary. Observations with fires were given a higher weight, and that weight was further scaled by the number of fires in the county. Here are the results of the model before weighting:

[4] https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json



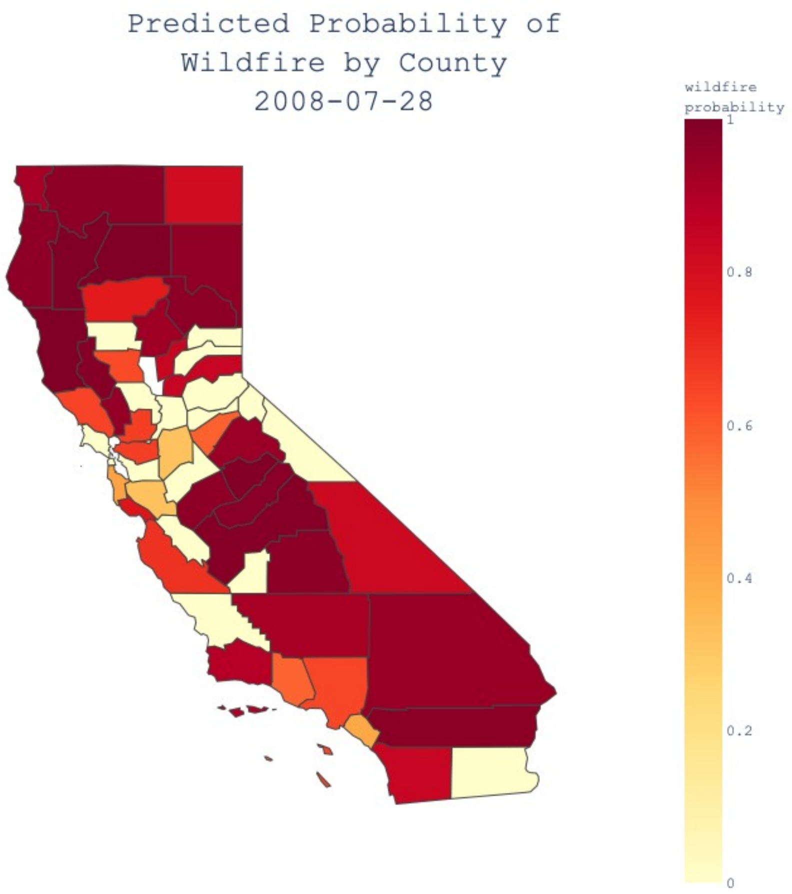
```
elser@fullpuff ~/p/M/randomForestFires> python3 ./runUnweightedModel.py
confusion matrix:
[[22813    66]
 [  621    59]]
sensitivity: 0.087, specificity: 0.997
PPV: 0.472, NPV: 0.974
accuracy score: 0.971
generating predicted probabilities for all data
```

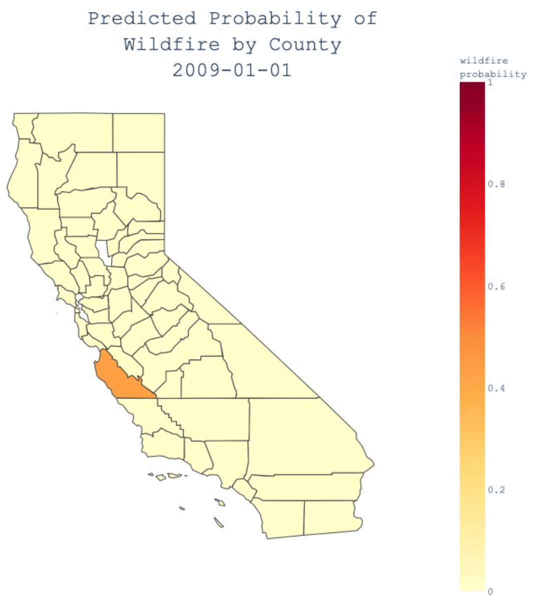
And after reweighting, the sensitivity improves nearly tenfold. Our positive predictive value decreases, however the cost of a false positive is extremely low in comparison to a false negative, so our gains seem more than worth it.

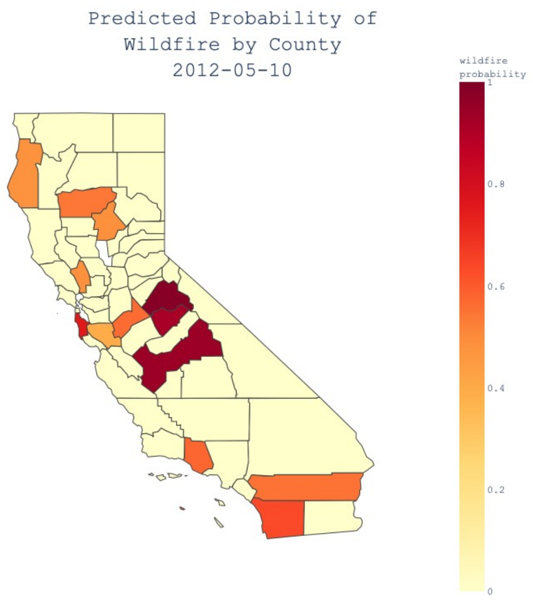


```
elser@fullpuff ~/p/M/randomForestFires> python3 ./runModel.py
confusion matrix:
[[18967  3912]
 [  109   571]]
sensitivity: 0.840, specificity: 0.829
PPV: 0.127, NPV: 0.994
accuracy score: 0.829
generating predicted probabilities for all data
```

Using this model, we can generate a predicted probability of having a fire in each county for each day, given the weather data. These predicted probabilities were generated for all per-county weather observations, allowing the creation of graphs like these:







Such graphs were created using `plotly`, which can generate choropleths using the generated data and geojson described above. Each observation was given a probability and graphed, which, when combined using `ffmpeg`, create an animation showing predicted probability of fire by county over time. This animation can be viewed at the github repository mentioned in the Requirements section, or can be generated (assuming the required data has been baked out by `bakeData.py` then `runModel.py`) by running `animPlotCounties.py` then `convertFramesToGraph.sh` .

4 Conclusion

In malesuada ullamcorper urna, sed dapibus diam sollicitudin non. Donec elit odio, accumsan ac nisl a, tempor imperdiet eros. Donec porta tortor eu risus consequat, a pharetra tortor tristique. Morbi sit amet laoreet erat. Morbi et luctus diam, quis porta ipsum. Quisque libero dolor, suscipit id facilisis eget, sodales volutpat dolor. Nullam vulputate interdum aliquam. Mauris id convallis erat, ut vehicula neque. Sed auctor nibh et elit fringilla, nec ultricies dui sollicitudin. Vestibulum vestibulum luctus metus venenatis facilisis. Suspendisse iaculis augue at vehicula ornare. Sed vel eros ut velit fermentum porttitor sed sed massa. Fusce venenatis, metus a rutrum sagittis, enim ex maximus velit, id semper nisi velit eu purus.