Summary

The model built predicts the specialities listed in `included_specialities` here (https://github.com/gupsydaisys/specialty_for_perscriptions/blob/master/Code%20for%20Physician%20Specialt short_path=5f5ee2b#L25) given a physcian's medicare pharma claims.

Although more specialities were in the data set provided, I chose to take out anything using the word `Nurse`, `Technician`, `Dentist`, and `Facility` since those specialities weren't about physicians and other that just didn't feel like they were a physician speciality eg `Physician Assistant`. With more time, I'd like to be (1) be more precise about which ones I choose since some of them I took out could have been a doctor specialties but I didn't have the time to look all of them up and (2) I'd like to map similar specialities to each other to clean up the data.

The big idea behind the model is to use how much a given physician perscribes each drug found in the data set as the features for the model: intuitively, within a specialty a physician probably perscribes the same proportion of drugs to their different patients regardless of patient panel size eg `Pain Management` specacilites probably perscribe a lot narcotics, a small amount of antihistamine, and not very much blood thinners. The biggest issue I see with doing this is some specialities that are highly surgical in nature might not ever be perscribing drugs to patients eg `Thoracic Surgery (Cardiothoracic Vascular Surgery).` I also would have liked to read some of the medical literature harder to identify specialities.

I made the assumption that I'd be given all the same medicare pharma claims without the `speciality_description` for a new doctor to classify.

Using my model and training on a set of 32_000 doctors, I was able to correctly identify the specialties of 4_856 out of 8_000 (`.607`). Given there were 83 specialities, `1/83 ~ .012`; I didn't do great but I did much better than random!

Modeling

Specifically, the model uses as features the porpotion of unique Medicare Part D beneficiaries with at least one claim (ie `bene_count`) for each possible generic drug seen in the dataset. Even more precisely, given there `n` generic drugs, there would be `n` features. For a given provider `p`, the `i`th feature is:

```
( min(bene_count, 5) for generic_drug i) / (total bene_count for provider p)
```

The dataset states that for `bene_count` that `[c]ounts fewer than 11 are suppressed and are indicated by a blank` hence so I replaced the blanks with 5 as the average between 10 and 0. I think 0 would have been a good option as well. I wanted the features with `NaN` values to have some weight just in case every generic drug perscribed was less than 11.

I also aggregated over `generic_drug` and not `drug_name` since I assumed that even if a doctor was perscribing to Patient A `Ventolin HFA` and Patient B `Proair HFA` given they are same generic_drug, they are probably trying to treat Patient A and Patient B in the same way.

Lastly, I chose `bene_count` instead of using `total_claim_count` or claims count based columns because I fully understood what the column meant and I felt like I'd be able to more confidently use it and manipulate it as such and the unique count made it so that it wouldn't be artificially high like `total_claim_count`.

I normalize over the total bene_count for that provider so that regardless of how many patients a provider sees the bene_count proportion should be around the same within the same speciality. One part I don't like about this is that total bene_count for a provider can have the same patient counted twice even the same patient uses two different generic drugs (a probably very common case).

Code & Performance

To view the associated code for the results, go (here) [https://github.com/gupsydaisys/specialty_for_perscriptions/blob/master/Code%20for%20Physician%20Specialt (https://github.com/gupsydaisys/specialty_for_perscriptions/blob/master/Code%20for%20Physician%20Specialt I would have liked to create functions out of some of the parts that I ran quite a few times and some unit tests.

The performance is very slow for building the initial model given you have `d` # of doctors and `r` # of times of rows associated with a doctor: the runtime is `O(d*r*r)` since you loop over all doctors and then loop over all generic drugs associated seen with the doctor and then for each generic drug you sum over the amounts every time you see it with the doctor. The space complexity if you have `d` # of doctors and `g` # of generic drugs is `O(d*g)`. When trying to predict the label for a new doctor, given `g` # of generic drugs and `r` # of times of rows associated with that doctor, the computation takes `O(r*r)` and space complexity is `O(g)`. I would have liked to use sql to build the features and labels since I didn't have time to process even all the data since my code ran so slowly.

Logistic Regression

Reading the problem statement, it looked like multi-class classification so I thought about using logestic regression, random forest and neural nets. I went with logestic regression because I understand how it works and I wanted to look at `theta` to understand if some drugs were more important in the classification than others. I also in sklearn can vary `C` the regularization term (which I didn't end up doing but would have liked to calibrate).

Modeling 2.0

If I had more time, I would have liked to create the following model. Instead of just having `bene_count` model, create in a similar way 12 models for: `bene_count`, `total_claim_count`, `total_30_day_fill_count`, `total_day_supply`, `total_drug_cost`, `bene_count_ge65`, `total_claim_count_ge65`, `total_30_day_fill_count_ge65`, `total_day_supply_ge65`, and `total_drug_cost_ge65`.

And then using predict_proba (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticR for each model and doctor, you'll have 12 arrays of 83 elements (specialities) where each element is the prob given the model and doctor that the doctor is that specialty. Then by taking the average or using each of them seperately train for sake of simplicity let's say you get another feature vector that has `96` values (`12*83 = 96`) and then use that feature vector to train another model. You then could look at theta to understand which initial model is best. I'm not sure if this really would produce better results but it was an interesting thought.

Results

My accuracy for predicting was `4_856 / 8_000 = .607` for 83 specialities and 32_000 doctors. I would have liked to run this a couple times.

An interesting question to ask is which specialities (if there is a systematic difference) did it do poorly at predicting. If so, were these surgical? Or were they not present very much in the data set? Or something else?