OKCupid Modeling Summary

This was my first unguided capstone for my Springboard course. I chose a dataset from OKCupid because I'm curious about relationships, unfortunately there wasn't much data that would reveal anything about relationships so I decided to see if I could classify members of the dataset as smokers and non-smokers. My use case was something like: What if you were using another dating service or even OKCupid itself and they provided a lot of information about a potential match, but that information didn't include whether or not they smoked. You could in theory use this model to predict if someone smoked or not.

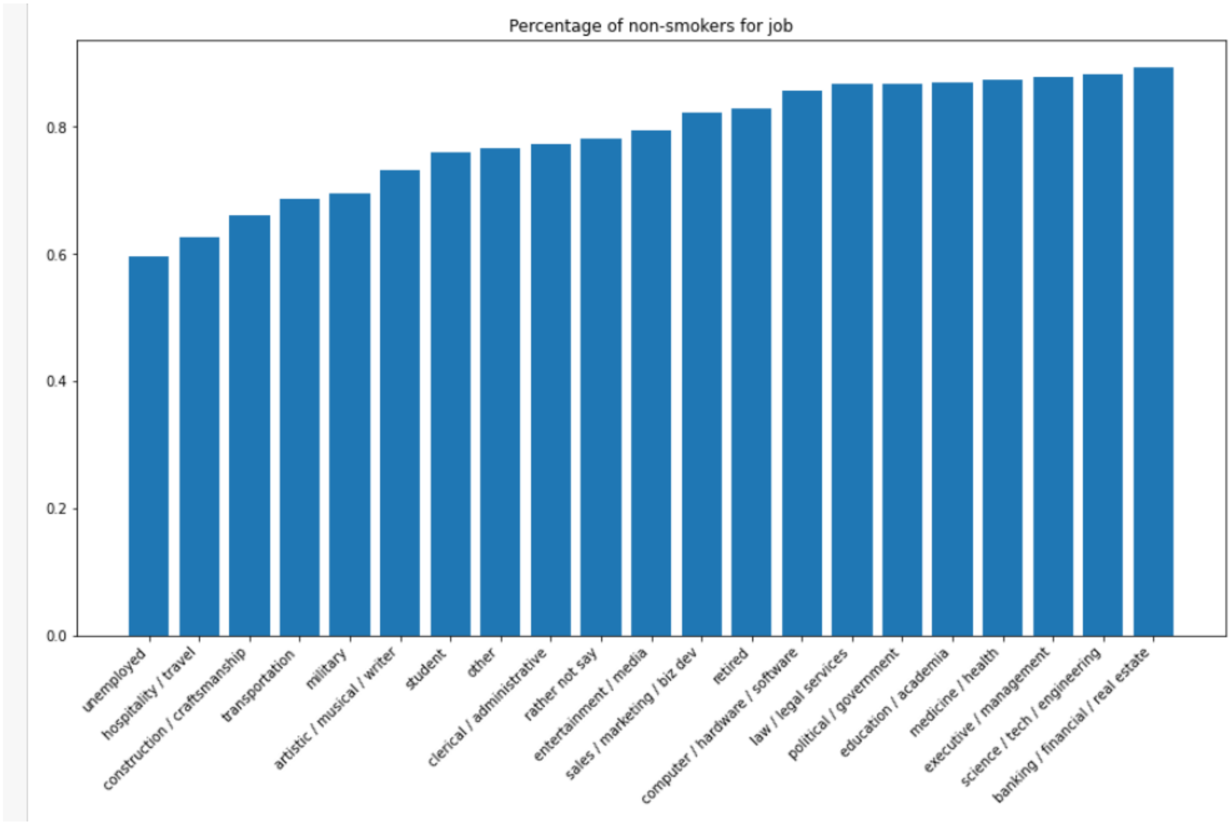
Features

Admittedly, I just tried to use all of the features I could. I figured if I had it available to me, why not try to use it? There were a number of dimensions I ended up having to cut because there were too many null values and no effective means of imputation IE: Offspring. Some features I dropped because there was so little variation that they were just clutter, IE: location, 99% of the dataset was from california.

I collapsed the astrological category from 46 categories encompassing folks signs and their attitudes about signs to simply their attitude. It was unnecessary clutter, and I've been given very little reason to give any weight to astrological signs

Both education and Employment stood out to me as significant correlates of smoking. Both a higher education, and higher prestige/income job were associated with lower rates of smoking. Ranging from 60% smokers for those who dropped out of highschool and all the way down to 5% smokers for those with a completed PHD. From 41% smokers for the unemployed all the way to 11% for finance

I've chosen to display non-smokers for the job category, as it's less dense than education and will be easier to consume. If the esteemed reader seeks to drill down farther, a jaunt over to the associated EDA jupyter notebook is recommended.



I created a few synthetic features. Largely as a result of trying to shape the data into a usable format. The column that indicated what language folks spoke was overloaded, it had a list of languages and a fluency score for each language. I broke this out into its own dataframe and added columns tracking the amount of languages folks spoke, and the total fluency score across all of the languages they spoke.

One of the synthetic features I created I might be suspicious of if I found it in the dataset underlying a model determining how much to charge folks for insurance or something. This synthetic feature is just ethnicity complexity. I added it mostly because it reused the code I had written for tracking multifluency. Given the structural racism present in our country feeding data like this directly into a model seems a little dangerous.

Optimization

There were a couple of different ways of optimizing the model available to me. I chose to optimize for Recall, because I liked the correlation to a medical context. Pretending I was analyzing test results to see if someone was sick made the whole exercise feel more meaningful. Recall can be defined as True positives divided by the sum of true positives plus False Negatives. Essentially, how many out of the total possible instances of the thing(Smokers in my case) did you catch?

You can apply recall to our dating scenario if you imagine that dating a smoker is an absolute deal breaker for you, and you don't even want a chance of being exposed to them. Well, optimizing for recall would enable you avoid any stinkers

I built a function that would do bayesian hyperparameter optimization on eight different models. Then I selected the model that was most effective. That model was generally most effective was the Gradient Boosting Classifier, capping out at about .64 for the recall score. I also had a Histogram Boosting Classifier that got as high as .51 recall score, The Histogram model runs substantially faster. Gradient takes about 97 seconds to train, whereas the Histogram model takes about half a second to train, so there's a possible use case for the Histogram model despite it's lesser recall score.

There are a few more things that could be interesting to explore. None of the models performed particularly fantastically. It would be interesting to explore if oversampling the dataset a bit enabled us to drive up the recall score somewhat. I would also be interested in setting up better logging for the bayesian hyperparameter tuning. It seemed like it was getting caught in local minima, as the optimal model would change from iteration to iteration. Finally there were four columns of answers to essay questions that I dropped, because at the time I didn't have the NLP skills to integrate them into the model. We didn't have the questions that the essays were answers to, but it's possible that simply vectorizing the essays would have enabled them to be profitably integrated into the model regardless.