Estimation of Warfarin Dose using Reinforcement Learning

Jason Lin

Motivation

Warfarin is a popular anti-clotting agent for blood. It is currently extremely difficult for doctors to determine the correct dosage of Warfarin for a specific patient, since the dosage can be very different for different patients. Furthermore, there can be dire consequences for giving a patient an incorrect dosage, so we are heavily incentivized to find a way to determine correct dosage amounts.

Currently, trial and error is used to determine the correct dosage for a patient, meaning they are first given a fixed dose, then that dose is adjusted based on their reaction to the dosage. This problem is related to reinforcement learning because we treat it as a multi-armed bandit, with “low”, “medium”, and “high” dosages being the three possible arms of the bandit.

I will use a linear bandit algorithm to predict the correct dosage of Warfarin to prescribe a patient in an online manner, without having to do a potentially dangerous trial and error beforehand.

Dataset and Algorithm

Our dataset consists of a set of 5700 patients and their respective features, as well as their ground truth effective Warfarin dosage.

I decided to use the LinUCB algorithm in order to determine the correct dosage amount for a given patient. This algorithm was chosen for several reasons.

Notably, this algorithm is incredibly computationally efficient, and can be done offline on previously recorded samples, which was needed in this case. Furthermore, since we also require sample efficiency, as we don’t have an infinite amount of mistakes we can make on real patients, the fact that LinUCB utilizes an Upper Confidence Bound and is an optimistic algorithm is extremely helpful. This means that it will achieve sublinear regret while also learning from each sample in order to minimize the number of patient samples needed.

Approach

I first determined two baselines: a fixed dose baseline, as well as the Warfarin Clinical Dosing Algorithm (linear baseline). I used these baselines to compare to my later results. I first implemented the LinUCB algorithm using the exact same features as the linear baseline. This approach beat the performance of the fixed dose baseline, but surprisingly, was still outperformed by the linear baseline.

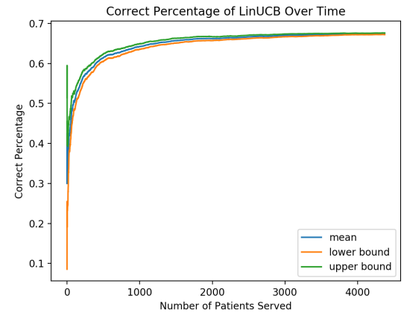
Thus, I decided to do feature engineering to create additional features to use in the LinUCB algorithm. After researching what useful additional features would be helpful, such as aspirin usage, genotypes, smoking, etc., and creating them, the algorithm was then able to beat the linear baseline.

Finally, for my further directions, I explored the usage of real valued reward functions and their effect on the percentage of severe errors in dosage assignment. I weighed the punishment of making a drastic error differently than the punishment for a small error. When the differences between the reward values were smaller, there didn’t seem to be much effect, but when I increased the punishment for severe mistakes, the algorithm was able to reduce its percentage of severe mistakes.

Additionally, I experimented with various values of the ridge coefficient in order to optimize performance of the algorithm, and found that 0.8 performed the best.

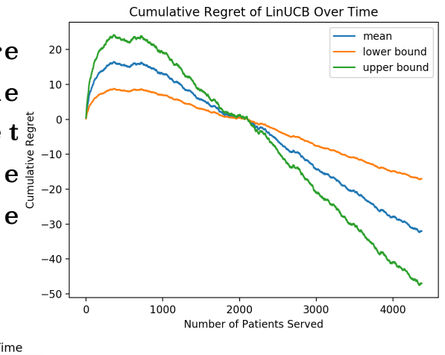
	Performance	Regret	% Severe Mistakes
Fixed Baseline	0.6117		
Linear Baseline	0.6513		
LinUCB 1st Attempt	0.6435	-70.9781	
LinUCB Augmented Features	0.6718	-128.7641	0.0036
LinUCB -0.5, -1 Rewards	0.6741	-32.0155	0.0043
LinUCB -1, -2 Rewards	0.6736	-68.6311	0.0027

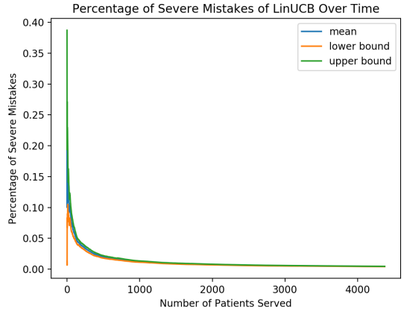
Results



These are results on LinUCB with augmented features, with a -0.5 reward for normal mistakes, and a -1 reward for severe mistakes.

The correct percentage steadily increases, the cumulative regret decreases, and the percentage of severe mistakes decreases.





The results are plotted with a confidence interval of 95%, averaged over a sample size of 20 random permutations of patients.

Future Work

According to my experiments, there was a significant improvement in the LinUCB algorithm when using additional engineered features. I would like to have more data on the patients, especially data related to factors that influence blood thickness, and I would like to see the effect that their inclusion has on the LinUCB algorithm. Furthermore, I’d like to know more about how various mistakes are weighted in order to fine tune and optimize the reward function further.