Estimation of Warfarin Dose using Reinforcement Learning

Jason Lin

Abstract

This paper uses a linear bandit algorithm, LinUCB, in order to determine the optimal dosage amount of Warfarin to administer to patients. It uses feature engineering to produce additional features that lead to an improvement in performance over the baseline. It also explores the use of real-valued rewards on both overall performance and a new metric, the percentage of severe mistakes that the algorithm makes.

1. Source Code

The source code for this project can be found at `https://github.com/jason2249/cs234`.

2. Introduction

Warfarin is a popular anti-clotting agent for blood. However, 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 initial 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, LinUCB, 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.

3. Background

We will treat our batch of patient data as if it were an online learning situation. This means at each time step, a new patient will arrive, we will observe his or her features, and then the LinUCB algorithm will make a decision on which dosage to use. It will then observe the reward from that dosage, and update its parameters accordingly.

We will consider 3 possible actions (arms of the bandit): a ”low” dosage, meaning under 21mg/week, a ”medium” dosage, meaning 21-49mg/week, and a ”high” dosage, meaning above 49mg/week. In our initial runs, we will give a reward of 0 if the correct dosage is prescribed, otherwise the reward will be -1.

3.1. Dataset

Our dataset consists of a set of 5700 patients and their respective features, as well as their ground truth effective Warfarin dosage. These features include patient information such as height, weight, age, race, and gender, among others.

I deal with missing data in two ways. With the features required for the baseline algorithms as well as the initial LinUCB algorithm, if the feature was missing, then I decided to remove that patient from the dataset and not count him or her entirely. My reasoning for this was that I felt adding in junk values for missing features would add in too much noise to the algorithm. Furthermore, after removal of all patients with missing features, there were still 4400 patients remaining in the dataset, and I felt that this was still more than enough data to train LinUCB on.

However, when I added augmented features to LinUCB, I noticed that these features contained a much higher prevalence of unknown or missing values. Thus, in this case, I decided to put a ”missing/unknown” value of 0 whenever a feature was missing or unknown in the dataset. This allowed me to maintain the dataset size of 4400 when training with augmented features.

4. Approach

4.1. Baselines

I developed two baselines for comparison to the LinUCB results later on. The first baseline was a fixed dose baseline, in which each patient who arrives is given a fixed dose of 35mg/week. The second baseline was the Warfarin Clinical Dosing Algorithm, which we will also refer to as the ”linear baseline”, since what it does is it uses a linear combination of select features from a patient in order to determine the op-

Algorithm 1 LinUCB

```
Input: ridge coefficient $\alpha$, patient features at time t $x_t$,
number of features $d$
for every time period t do
    Observe patient features $x_t$
    for each arm do
        if arm a hasn't been seen then
            Initialize $A_a$ as a d x d identity matrix
            Initialize $b_a$ as a d x 1 0 vector
        end if
        $\hat{\theta}_a = A_a^{-1} b_a$
        Payoff = $x_t^T \hat{\theta}_a + \alpha \sqrt{x_t^T A_a^{-1} x_t}$
    end for
    Choose the argmax arm $a*$ over all arm payoffs
    Observe the reward $r_t$
    $A_{a*}$ += $x_t x_t^T$
    $b_{a*}$ += $r_t x_t$
end for
```

timal dose for a patient. In particular, the features used were patient age, height, weight, race, enzyme inducer status, and amiodarone status.

4.2. Initial LinUCB

To improve upon the baselines, 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[1], 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.

The LinUCB algorithm is outlined in **Algorithm 1**.[2]

My first implementation of the LinUCB algorithm used the exact same features as the Warfarin Clinical Dosing Algorithm baseline. This approach beat the performance of the fixed dose baseline, but surprisingly, was still outperformed by the linear baseline.

4.3. Augmented LinUCB

In order to improve upon my initial LinUCB implementation, I decided to use feature engineering to augment the original implementation with additional features. I looked at what available features were in the provided patient dataset and researched them and what their effects were. Since I knew that Warfarin was an anti-clotting agent for blood, I specifically looked for patient characteristics that would have an impact on blood thickness or thinness.

I noticed that whether or not a patient was a smoker was a feature that was provided. According to smokefree.gov, smoking makes blood thicker[3], so I thought this would be a good feature to include in my augmented LinUCB algorithm, as presumably, a patient who smokes would need to have a higher dose of Warfarin in order to compensate for their thicker blood. Thus, I added whether or not a patient was a smoker as a feature in my augmented LinUCB algorithm

Additionally, I noticed that whether or not a patient was currently taking aspirin was a feature that was provided. According to fda.gov, aspirin is a blood thinner[4], so I thought this would also be a good feature to include in augmented LinUCB in addition to smoking, as presumably a patient who is taking aspirin would need less of a Warfarin dosage assigned to them, as their blood would already be thinner from the aspirin. Thus, I added whether or not a patient was taking aspirin as a feature in my augmented LinUCB algorithm as well.

Other additional features that I included were gender, CYP2C9 genotype, and VKORC1 genotype. I decided to include the CYP2C9 and VKORC1 genotypes of the patient since I noticed that these genotypes were used as features in the second provided linear baseline, the Warfarin Pharmacogenetic Dosing Algorithm, so I thought they would be helpful to include in my augmented LinUCB as well.

Upon updating my algorithm with these augmented features, I was able to beat both the fixed dose and linear baseline performance.

4.4. Real Valued Rewards

I also explored the usage of real valued reward functions. In order to better see the benefit of experimentation with reward functions, I created another evaluation category: the percentage of severe errors that the algorithm makes. I classified a "severe" error as an error that was 2 or more dosage levels away from the actual dosage. For example, a severe error would be when a patient's actual dose was supposed to be low, but we instead assigned them a high dosage, or vice versa.

Explicitly, what I did to make the reward functions real values is I weighed the reward of making a moderate error differently than the reward of making a severe error. For one example, the reward for making a moderate error was -0.5 and the reward for making a severe error was -1, instead of -1 on all errors. The hope for this reward scheme was to decrease the percentage of severe errors that occurred.

In my experimentation, I found that when the real valued rewards for moderate and severe mistakes were closer together in absolute value, then the reward function had little to no effect on the percentage of severe mistakes. However, I found that when I more heavily punished severe mistakes by increasing the absolute difference between the two mistake types, the algorithm was able to successfully decrease the percentage of severe mistakes it made.

4.5. Ridge Coefficient

Additionally, I experimented with various values of the ridge coefficient in order to optimize performance of the algorithm. Through careful hyperparameter tuning, I was able to find that a ridge coefficient of 0.8 generally performed the best for my LinUCB algorithm.

5. Experiment results

5.1. Description of Result Statistics

I kept track of 3 major statistics: the percentage of correct dosage recommendations, the cumulative total regret, and the percentage of severe mistakes that the algorithm made. I will now briefly describe each statistic.

The percentage of correct dosage recommendations is fairly straightforward. It is simply the number of correct dosage recommendations (bucketed in terms of ”low”, ”medium”, or ”high” dosages) divided by the number of total dosage recommendations.

The cumulative regret is given by the equation:

$$R_T = \sum_{t=1}^{T} E[max_j[X_t^T B_j] - X_t^T B_i]$$

Where X_t is a patient's feature vector, and B_i and B_j are parameter vectors.

There is a different beta parameter vector for each arm. The intuition behind the $max_j[X_t^T B_j]$ term is that we will multiply the patient's feature vector by the beta vector for each arm. This will return an expected reward for each arm, and we want to take the maximum reward out of all arms as the target value for our regret. We then subtract the expected payoff of the chosen ”best” arm that our algorithm chose from this target value to determine regret at a certain time step t.

We determine the beta parameter vectors beforehand by running a batch linear regression for each arm. For every single patient in the dataset, we assign them a value of either 0 or -1 based on what their correct dosage is for that arm. For example, if a patient's correct dosage is less than 21mg/week, then for the labels of the ”low” arm linear regression, we will label the result as 0. In the ”medium” and ”high” dose linear regressions, we label the result for this patient as -1, since the medium and high dosages are wrong for this patient.

Table 1. Results of correct dosage percentage, cumulative regret, and severe mistake percentage on various models.

Model	% Correct	Regret	% Severe
Fixed Baseline	0.6117	119.085	0.0
Linear Baseline	0.6506	76.229	0.0011
LinUCB 1st	0.6387	42.553	0.0034
LinUCB Aug.	0.6716	76.931	0.0042
LinUCB -0.5, -1	0.6736	106.646	0.0048
LinUCB -1, -2	0.6735	67.317	0.0030
LinUCB -1, -4	0.6717	64.912	0.0024
LinUCB -1, -8	0.6566	79.404	0.0021

After training these batch linear regressions and determining the beta parameter vectors for each arm, we use these beta parameter vectors in the calculation of total regret, according to the equation given previously.

Finally, we tracked the percentage of severe mistakes that the algorithm makes. As mentioned previously, this was added to track the effect of changing the reward functions to give real valued rewards. Like the percentage of correct dosages, this statistic is also fairly straightforward, as it is given by the number of ”severe” mistakes that the algorithm makes divided by the number of total dosage recommendations the algorithm makes.

5.2. Correct Percentage, Total Regret, and Severe Percentage Final Results

The final results for correct percentage, total cumulative regret, and severe mistake percentages are highlighted in **Table 1**. The results as a function of the number of patients who have arrived over time are shown in **Figure 1**, **Figure 2**, and **Figure 3**.

From the results, we can see that the fixed baseline had the worst performing correct diagnosis accuracy by far, which is not unexpected. It also had the highest cumulative regret. However, since it only ever assigns a medium dose, it is impossible for the fixed dose baseline to make a ”severe” mistake, under our definition of severe. This means that if we wish to completely prioritize not making a severe mistake, then the fixed dose baseline is actually the optimal algorithm!

The linear baseline is able to improve upon the fixed baseline in both the correct percentage and the cumulative regret, but we begin to see some severe mistakes occurring. This may be a good choice for those who wish to have very little severe mistakes occur, while still having higher dosage accuracy

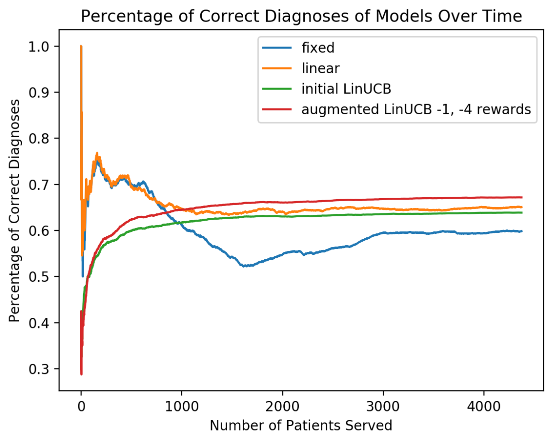

Figure 1. The percentage of correct diagnoses that various models made, as a function of the number of patients that have arrived.

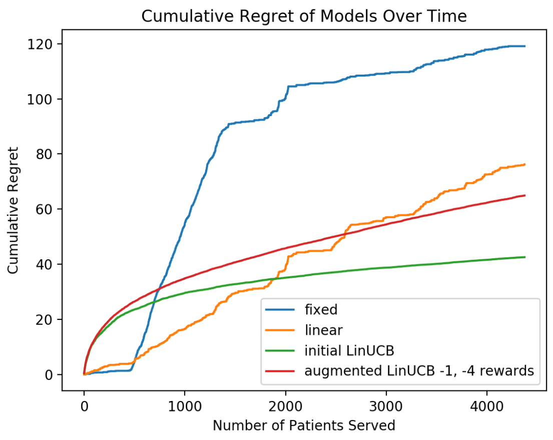

Figure 2. The cumulative regret of various models, as a function of the number of patients that have arrived.

and lower regret.

Interestingly, the first attempt at the LinUCB algorithm beat the correct percentage of the fixed baseline, but was unable to beat the fixed percentage of the linear baseline. On the other hand, it was able to achieve the lowest regret out of all models! Since this was unable to beat the performance of the linear baseline, I decided to pursue LinUCB further by augmenting its features, as mentioned above.

When run with augmented features, the LinUCB algorithm was able to beat both the fixed and linear baselines in correct percentage, and it was able to match the linear baseline in cumulative regret. However, I noticed that the percentage of severe mistakes was much higher than that of all previous runs, which could be a major problem if a doctor would like to avoid as many severe mistakes as possible.

For this reason, I decided to pursue the direction of real valued rewards instead of simply 0 and -1, which I elaborate on further above. My first attempt at changing the reward function lead me to give a reward of -0.5 for a moderate mistake, and a reward of -1 for a severe mistakes. This actually lead to an increase in regret and severe mistake percentage, while the correct percentage stayed the same. Thus, this model did not produce very useful results.

Next, I decided to make the punishment of making a severe mistake worse by decreasing the rewards to -1 and -2 for moderate and severe mistakes respectively. Here, I saw fruitful results, as the regret and severe mistake percentage both dropped to below the levels of the initial augmented LinUCB model, while the correct percentage remained above that of the initial LinUCB. I decided to pursue this idea further by further increasing the punishment for making a severe mistake.

I tried out the algorithm with a reward of -1 for moderate mistakes and both -4 and -8 for severe mistakes. I received the best results on the -1,-4 reward model, as the correct percentage of this model still beat the initial augmented LinUCB model, and it had the lowest regret and severe mistake percentage. The model with -1,-8 rewards began to get worse, as the correct percentage dropped and the regret started to increase. However, the severe mistake percentage continued to drop, which tells us that if we keep increasing the punishment for making a severe mistakes, we can keep pushing the percentage of severe mistakes lower if needed.

5.3. Confidence Bound Graphs

In order to ensure that my results were accurate and not due to the random ordering of the patients, I ran the augmented LinUCB model with -1,-4 rewards on 20 different permutations of the patients, and plotted the average of my results, as well as 95% upper and lower confidence bound. The resulting graphs can be seen in **Figure 4**, **Figure 5**, and **Figure 6**.

6. Conclusion

According to my experiments, there was a significant improvement in the LinUCB algorithm when using additional engineered features. In the future, 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. I believe this would greatly strengthen the predictive power of the algorithm, as we already saw how adding just a few

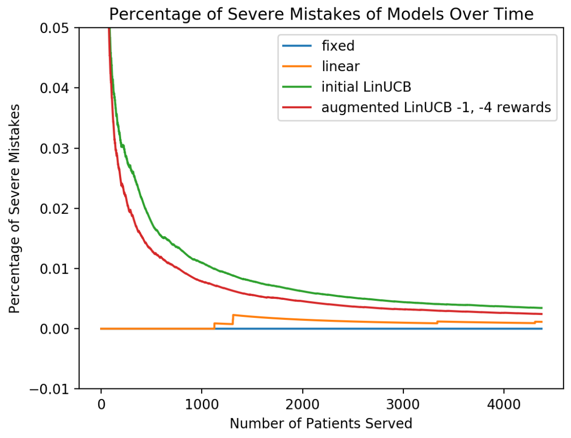

Figure 3. The percentage of severe mistakes that various models made, as a function of the number of patients that have arrived.

more blood thickness related features was able to boost the performance of the 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. In this problem description, we are currently only given whether a dosage is incorrect or correct. However, I would like to know if, for example, higher dosage mistakes are more favorable than low dosage mistakes, or vice versa. This would allow me to make the reward function reflect such values, and thus would make this algorithm much more valuable to doctors when they actually use it!

7. References

1.https://arxiv.org/pdf/1003.0146.pdf
2.http://john-maxwell.com/post/2017-03-17/
3.https://smokefree.gov/quit-smoking/why-you-should-quit/health-effects
4.https://www.fda.gov/drugs/safe-daily-use-aspirin/using-aspirin-lower-your-risk-heart-attack

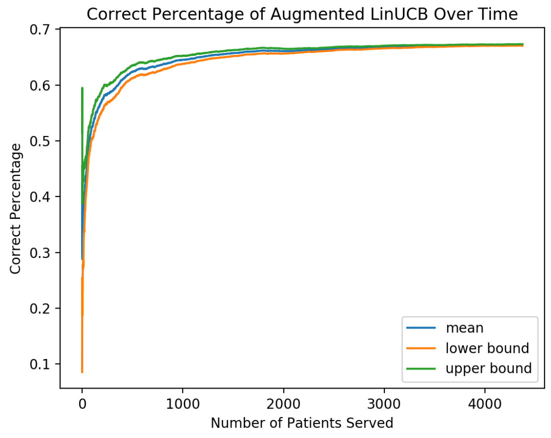

Figure 4. The percentage of correct diagnoses, along with upper and lower confidence bounds, that the Augmented LinUCB model with -1,-4 rewards made, as a function of the number of patients that have arrived.

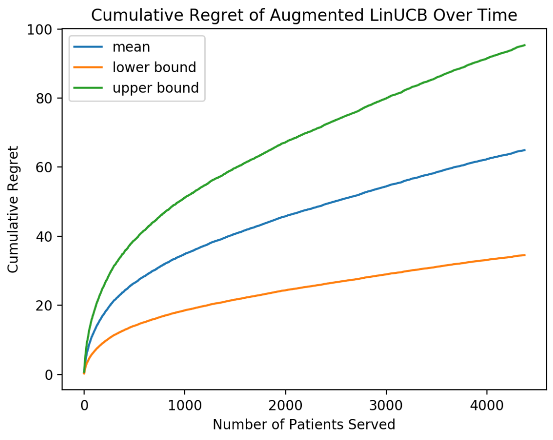

Figure 5. The cumulative regret, along with upper and lower confidence bounds, of the Augmented LinUCB model with -1,-4 rewards, as a function of the number of patients that have arrived.

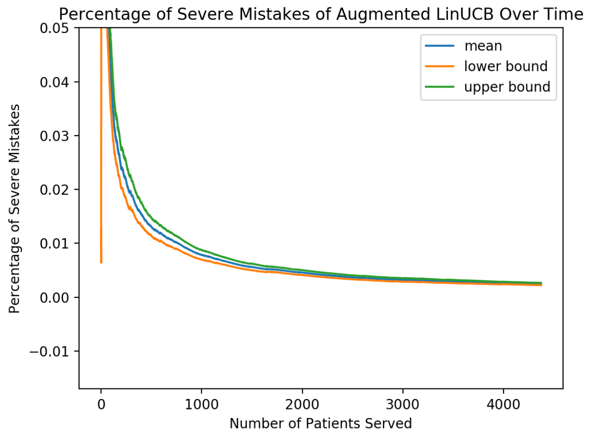

Figure 6. The percentage of severe mistakes, along with upper and lower confidence bounds, that the Augmented LinUCB model with -1,-4 rewards made, as a function of the number of patients that have arrived.