Problem 1

Briefly explain how you used batch stochastic gradient descent with regularization to learn the weights. Think about how the regularization is incorporated into the loss function and how that affects the gradient when updating weights.

The way that we implemented batch stochastic gradient descent with regularization is that when we were training our model, we separated our dataset into certain batches where we would compute a loss function for each one of those batches which in this case we used binary log loss where negative logarithms are taken to calculate the loss. As the name suggests, a model with a lower loss function will be one that performs better. For each one of those batches, we would then add a regularization term when we are computing the gradients. This is because our $\lambda > 0$ which means that the regularization term that will be added will make the gradient larger where the extent depends on the size of the hyperparameter. The main purpose of this regularization term is to reduce the chance of our model overfitting to the training data. The reason why regularization works is that it reduces the size of our weights to which could lead to less bias and overfitting possibility since our weights are more generalized. The regularization is incorporated into the loss function through L2 regularization based on the handout where we will square the value of our weights and then tune with our λ hyperparameter. This affects our gradient because when this is added to our gradient and then subtracted from our weights, this will lead to more generalized, smaller weight values which will improve our overall performance of our model. Therefore, this is how I incorporated regularization inside of this training model.

Problem 2

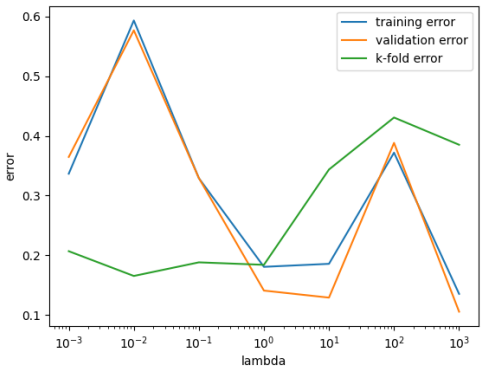
Think back to when you implemented Logistic Regression on the Census dataset. How would it have been different if you applied Tikhonov regularization? Specifically, how would the regularization affect the accuracy and the types of errors?

As we discussed in the previous problem, the whole point of regularization is to reduce the complexity of the model to try to prevent overfitting. The way that Tikhonov regularization, or L2 regularization, accomplishes this is by creating a regularization term that is added to the loss function which will decrease the value of the weights to make them more generalized to prevent from high weight values to prevent from potentially overfitting on our training dataset. Therefore, if Tikhonov regularization was implemented in our logistic regression model on the Census dataset, this this would have made our model less complex as our weight values would have been more regularized. This will lead to a decrease in estimation error because from the bias-complexity theorem, if the complexity decreases, then the estimation error decreases as well. However, we don't want this to play into our accuracy because if we tune our hyperparameter λ too much, then this could potentially lead

to underfitting and eventually lower accuracy and a potential higher approximation error. Therefore, while regularization could be implemented to reduce the estimation error, we need to be careful to balance the bias-complexity tradeoff by not tuning our hyperparameter too much because this could lead to lower accuracy and a higher approximation error due to underfitting if we are not careful.

Problem 3

Use plotError(), which we have implemented for you, to produce a model selection curve. Include your plot here. Then, conclude what the best value of lambda is and explain why. NOTE: It takes about five minutes to generate a graph. Please set your default lambda in the constructor to your optimal lambda you discovered for TA testing purposes.



In the graph above that came out from my plotError(), it seems to be that the graph's shape is high on the left and right sides for the training error and the validation error, and it seems to be the lowest in the middle of the graph. Therefore, it seems to be that the optimal value of lambda is around 10^1 where I was able to set my default lambda to be equal to 10 which I was able to get the highest accuracy with. From our lecture notes, we know that if our training and validation error start out to be high, then this is a sign of underfitting, so choosing a lambda value near the left-side of the graph wouldn't be the most optimal. Using the same logic, as the model gets model complex as you go to the right hand side, we can see the validation error peaking again which is a sign of overfitting. Therefore, the sweet spot seems to be in the middle which our graph suggests is around 10^1. It also makes sense to choose the lowest validation error as the spot where the optimal lambda should be because the whole point of the validation dataset is to avoid overfitting. If we didn't have this validation dataset, then if we evaluate multiple hypotheses on the test set, and then pick the best one, then it is no longer an unbiased estimate of $L_D(h)$ because in a way, we are trying to overfit our test dataset, but once again, it wouldn't generalize to another test set. Therefore, a validation dataset will will be a way to prevent overfitting since we haven't

tested on this validation dataset before, so this would be the error that I would most highly value. For the k-fold error, we can also see that after the lambda values get increasingly positive, the error skyrockets up which is a sign of overfitting as well since our model gets more complex, so this is another sign that our optimal lambda is near the middle of the graph. Therefore, all in all, the sweet spot for our lambda value seems to be around the 10^0 to 10^1 region, so since the validation error dips the lowest around $\lambda = 10$, this is the final optimal lambda value that I chose for my model.

Problem 4

In this project, you used validation data to select a model. Suppose that each patient might've had multiple samples (e.g., multiple lab tests or x-rays) collected and entered into the dataset. Would you need to account for this when splitting your train-validation-test data? If yes, how? If no, why not? (3-5 sentences)

Yes, it would be important to account for this when splitting train-validation-test data because if a single person has multiple different samples, and if they are separated into different sets (i.e., one person's samples are in both the training and validation datasets), then this could lead to overfitting for two reasons. First, since our samples are not identically and independently distributed (since two samples could come from the same patient), this goes against our assumptions for our model for a fixed hypothesis class which is definitely something that we should be taking into account. Secondly, since once the model is trained on the training data on a specific patient, it would bleed over to the validation dataset, so the model wouldn't be testing the validation set over new patients, but it could recognize that the sample inside of the validation dataset comes from the same patient from a sample in the training dataset. This means that whenever our model is being tested, it is not generalizing to new patients. So, it is important when splitting our train-validation-test data that we don't allow the samples of one patient to be split up into different datasets. By not allowing the samples of a patient to be split up, this will ensure that our model will be testing on new patients instead of potentially overfitting by recognizing that the samples from the same patient are in both the training and testing datasets.