Phys 2620J Homework 3

Andrew Friberg
(Discussed with Liam Storan and
Joseph Cavanaugh)

Problem 1

Part A and B:

The first order of business is to find the probability that you will win a single round of *Gambler's Ruin.* Assume that your friend rolls her dice and gets a value m. We win if we roll a value $n > m$, or if $n = m = 1$, or if $n = m = 6$. We could lay out all the possibilities in a 6 by 6 grid, but we can also calculate it as shown in the table:

m	$P(m)$	$P(n > m)$	$P(win\|m)$
1	1/6	5/6	1
2	1/6	2/3	2/3
3	1/6	1/2	1/2
4	1/6	1/3	1/3
5	1/6	1/6	1/6
6	1/6	0	1/6

We can calculate the total probability of winning the round: $p = P(win) = \sum_{i=1}^{6} P(win|i) \times P(i) = \frac{17}{3} \approx 47.22\%$. The probability of losing is $q = 1 - p \approx 0.5278$. Just from this, it seems that we have a decent chance of winning some money in this game, but let's continue calculating. Define U_k to be the probability of winning *Gambler's Ruin* assuming you start with k dollars. For $k = A = \$60$ we see trivially that $U_A = 1$ since we already started out in the winning situation. Similarly, $U_0 = 0$ since we've already lost. We established in lecture that

$$-q * U_{k-1} + U_k - p * U_{k+1} = 0$$

We can write this in matrix form. I coded this into excel and transferred it into matlab (I may include a screenshot of my vector at the end of this homework), to find that $U_{50} \approx 0.328$. This means that our expected cash value after playing this game are $\mathbb{E} = \$0 \times (1 - 0.328) + \$60 \times 0.328 = \$19.68$ and our expected loses are $\$19.68 - \$50 = -\$30.32$

In most cases, our "friend" is earning much more money from this game than we are.

Part C:

If the earnings could take any value, for instance if we could vary the amount that we bet (and hence have a chance of winning) then we'd have a case where we can move directly to other $U_k's$. The conditional probabilities calculated in lecture would not be as simple as before. For instance, I believe (although I haven't explicitly calculated every step) that the general form of a game where you can bet anywhere between \$1 and M dollars would take the form

$$U_k = \sum_{i=i}^{M} [q_i \cdot U_{k-i} + p_i U_{k+i}]$$

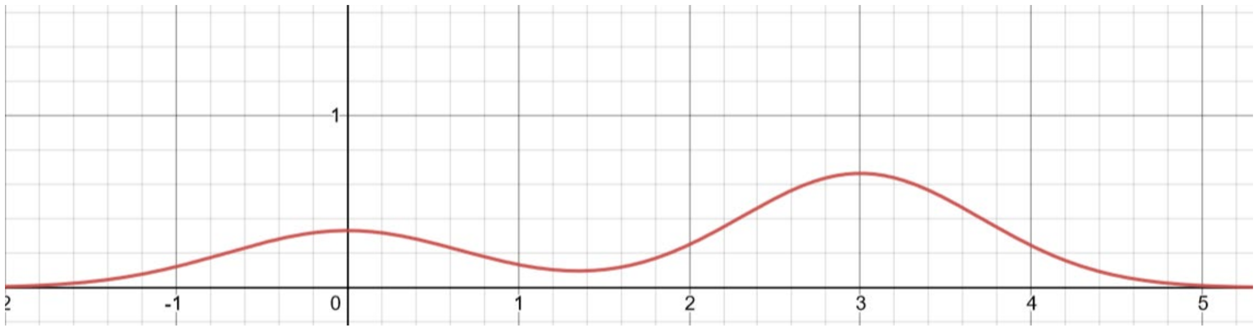
Where p_i and q_i are the probabilities of winning or losing i dollars, respectively. We then would get a much more dense matrix that we could invert using the conditions that $U_A = 1$ and $U_0 = 0$

However, there is also a messy element of game theory involved in this that I'm not sure how to account for, because we choose at the start how much money to bet. If $p_i = p_j = 1 - q_i = 1 - q_j \quad \forall i, j$ then I think a better strategy is actually to bet everything you have at the beginning. Because the odds are stacked against you, every step that you delay you have more probability of losing. Assuming the odds are the same as in parts a and b, this would give you an expected winnings of $(\$0 \cdot q + \$60 \cdot p) - \$50 = -\$21.668$ which is higher than the expected winnings of part b. However, the best strategy is still just not to play.

Problem 2

I had many issues with my code in this question, so there are two files associated with it

Part A:

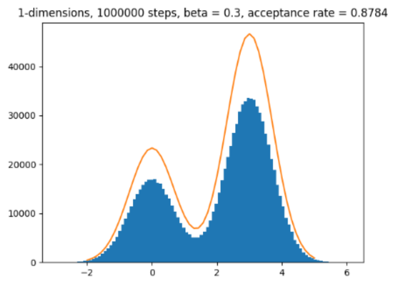


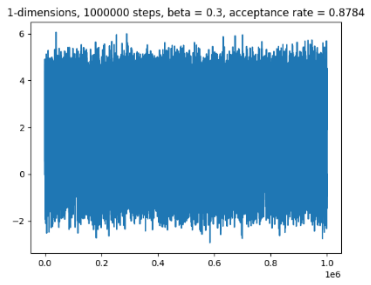
Plotted on Desmos

Part B: All coded

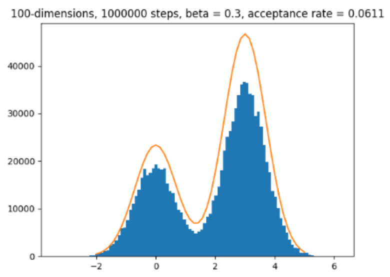
Part C:

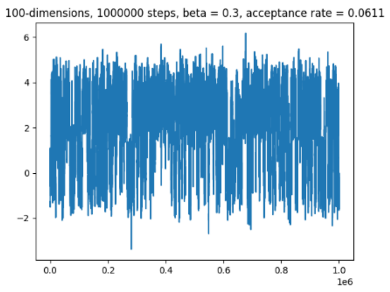
The trace plots are all contained in a linked directory, the histograms for the 1-dimensional case is shown below:





The acceptance rate is very high, the distribution closely resembles what we got in part a[1], with the trace plot showing that it explores the space very evenly. Contrast this with the 100 dimensional case:





[1] At this point in my coding, I was not yet aware that plt.hist had an optional parameter density=True which would have improved my life, but all of this code was written before I was aware of such a useful parameter

The histogram is much more jagged than for the one-dimensional case, and the acceptance rate is almost ten times lower. The trace plot reflects this, as our MH algorithm proposes a lot of steps with a really low transition probability. We see two “regions” in the trace plot, one for each peak of the probability distribution. Generally, as the dimension increases, the acceptance rate plummets, because the distribution is now higher dimensional, which means more pronounced probability peaks and forces the steps to be smaller. Keeping beta constant means that our algorithm doesn’t work extremely well.

Part D:

Dimension	1	2	3	4	5	70	80	90	100
Acceptance Rate	0.8784	0.8204	0.7666	0.7334	0.6871	0.12	0.1082	0.0802	0.611

Part E:

I tried automating this, but my algorithm kept unexpectedly and frustratingly crashing. I ended up doing it by hand, the results of which are in *finding_beta()*. The data is summarized below:

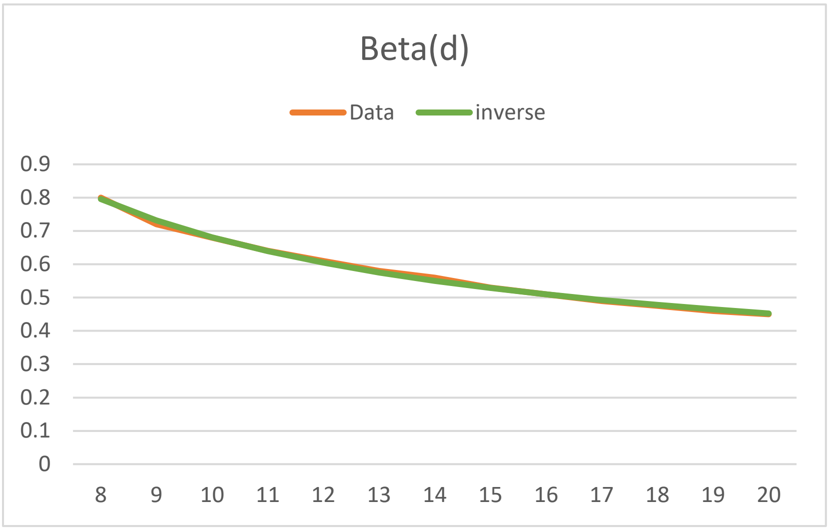
dimension	8	9	10	11	12	13
beta	0.8	0.72	0.68	0.64	0.61	0.58

dimension	14	15	16	17	18	19	20
beta	0.56	0.53	0.51	0.49	0.475	0.46	0.45

The fit that I found worked well for the $d \in [8,20]$ range was

$$\beta_d = 0.223 + \frac{4.579}{d}$$

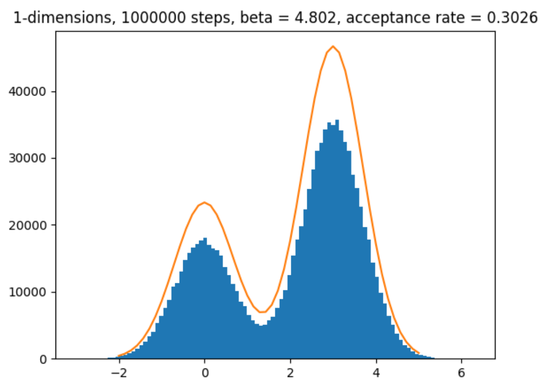
This was found in the “Beta Fitting.xlsx” spreadsheet. This function didn’t seem to keep the acceptance rate from dropping significantly when going into higher dimensions, but the fit worked really well for the dimensions on which I fit it.[2] The data and line of best fit are shown in the graph.

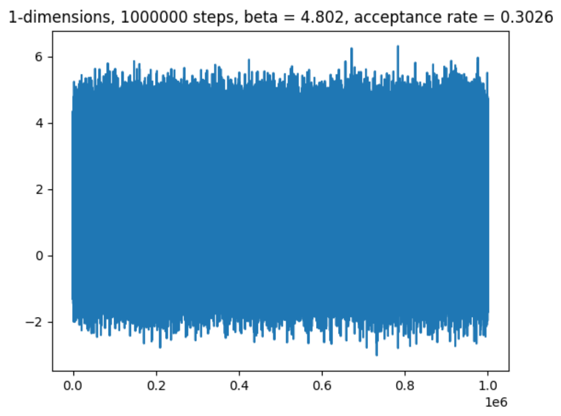


[2] I think I remember in Office Hours hearing that a logarithmic function was expected. That unfortunately was not what I observed, and it would be pretty disingenuous to say otherwise :(

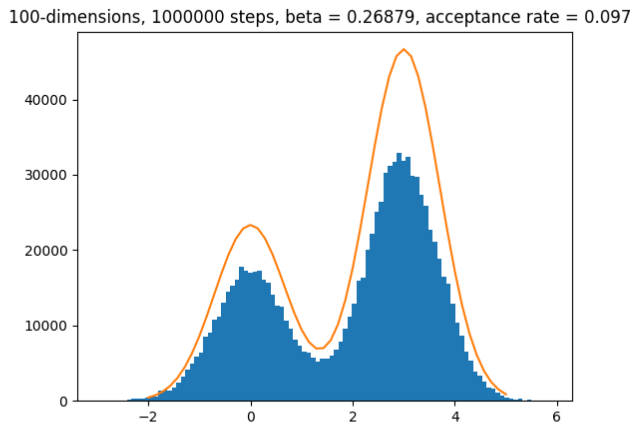
Part F:

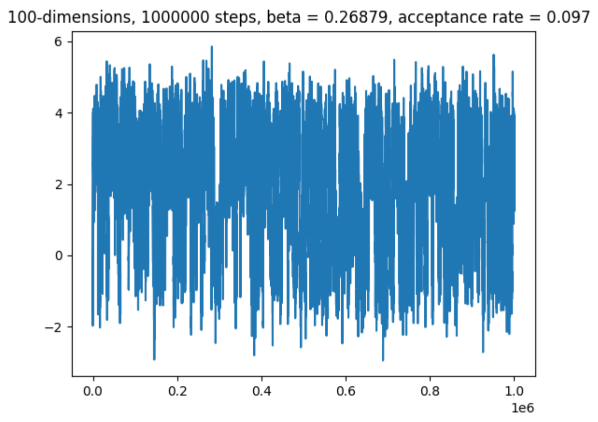
All of the trace plots are contained in an attached folder, but we'll focus on dimensions 1 and 100 here so that we can compare. A one-dimensional histogram and trace plot are shown below:





Once again, the distribution closely resembles what we would expect, and the trace plot shows a uniform spread. The acceptance rate is also much closer to 0.234 than previously, indicating that the fit we found for beta works for some dimensions.

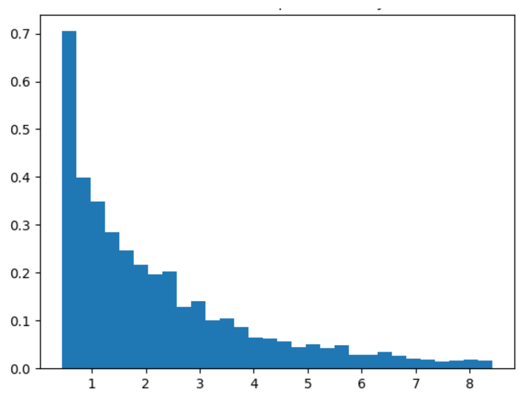




The histogram looks somewhat less jagged than the histogram we used when we didn't adjust beta. The trace plot looks a bit more evenly distributed, with much less of a division between the upper and lower halves, indicating that the algorithm is moving more efficiently between the higher and lower peaks. The acceptance rate is not ideal, meaning that we would need to examine the behavior of beta at high dimensions to create a function that would get us to the 0.234 acceptance rate that we are looking for.

Problem 3

Part A:



The horizontal axis is the lifetime in microseconds (I used microseconds as my basic unit throughout this problem) and the vertical axis is the probability density of muons which fell into that time.

The full probability density function (with an explicit normalization constant) is

$$p(t) = \frac{e^{-\frac{t}{\tau}}}{\tau\left(e^{-\frac{t_0}{\tau}} - e^{-\frac{t_s}{\tau}}\right)} = A \cdot e^{-\frac{t}{\tau}}$$

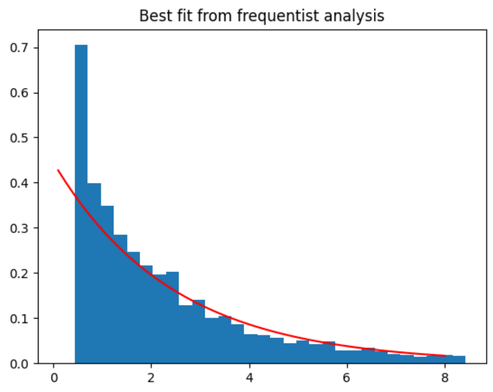
With $t_0 = 0.1\mu s$ and $t_s = 8\mu s$

Part B:

We can find the joint likelihood function by multiplying the probability of observing each datapoint given a certain lifetime. Given n instances of data t_n collected in the database D, we compute the likelihood as

$$\mathcal{L}(\tau|D) = \prod_{i=1}^{n} p(t_i) = A^n \exp\left(-\frac{1}{\tau}\sum_{i=1}^{n} t_i\right)$$

In the frequentist framework, we can maximize this likelihood (Maximum Likelihood Estimation) to find the best estimator of τ. Using scipy.optimize (you can also calculate it analytically) we find that $\hat{\tau} = 2.436\mu s$. This curve of best fit is plotted below, together with the data.



Part C:

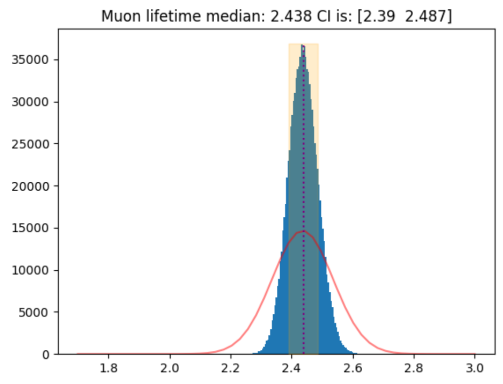
Baye's rule is as follows:

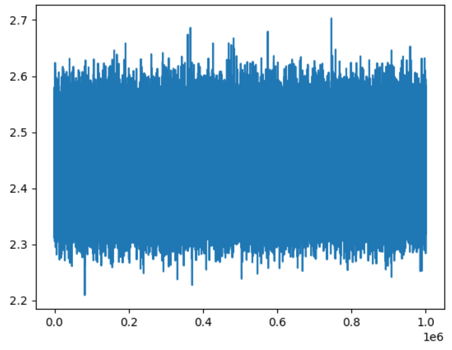
$$P(\tau|D) = \frac{P(D|\tau)\cdot P(\tau)}{P(D)}$$

Using this equation, we can take some prior best guess for the parameter, multiply it by the likelihood function $P(\tau|D) = \mathcal{L}(\tau|D)$, and find the posterior distribution and find the region in which most of the density lies to find confidence intervals for our data. Since we can't calculate $P(D)$ directly, we can use MCMC – MH to approximate this distribution function that we don't know the shape of.

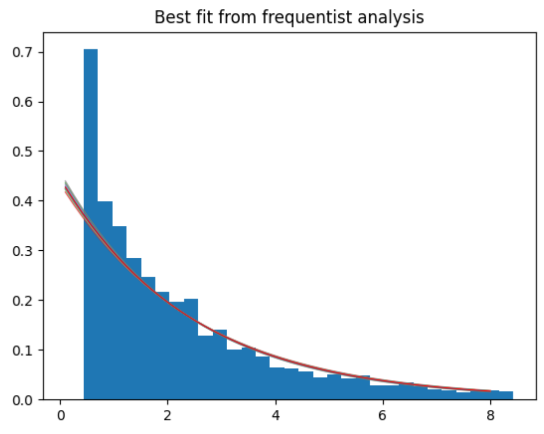
Part D:

The results of implementing MCMC – MH for the unknown Bayesian formula gives us the following distribution and trace plot:



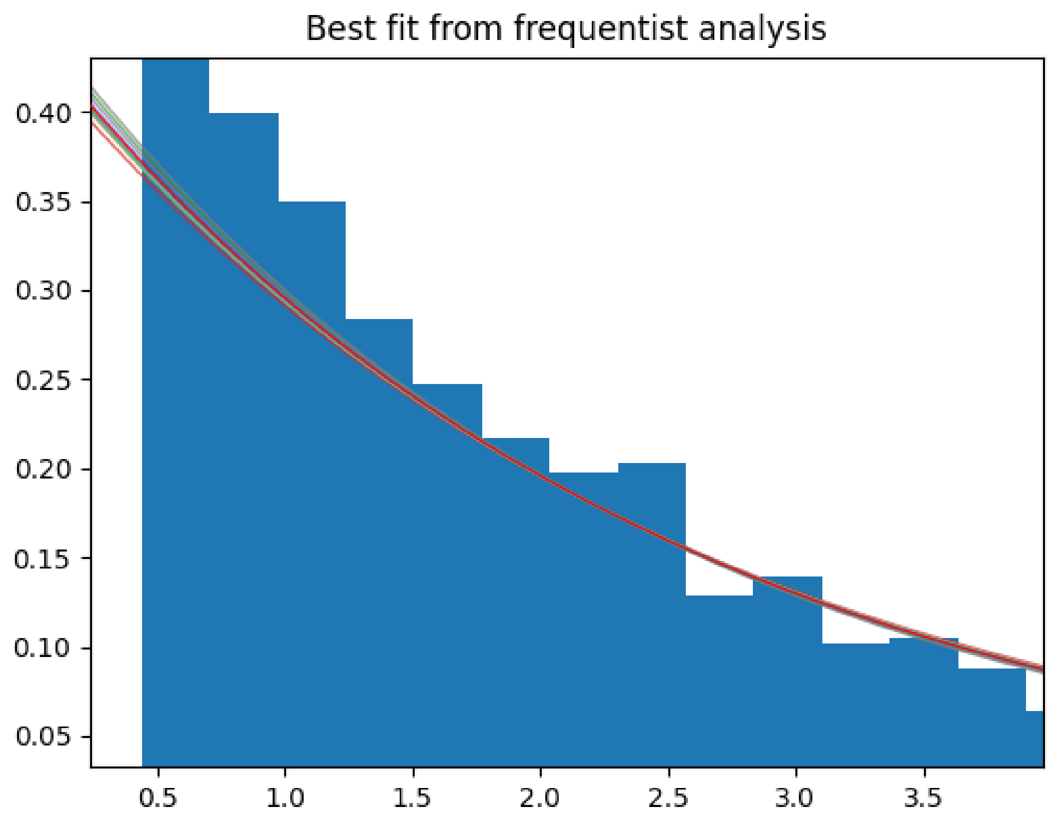


(The height is not relevant, I just forgot to specify density=True) This was run with 10^6 steps, with a gaussian proposal distribution centered at 0 and with standard deviation of $1\mu s$, which gave an acceptance rate of 22.3% and we can see from the trace plot that the algorithm has converged. The median (2.438 microseconds) is shown as a purple dotted line, and the 68% confidence interval is $[2.39, 2.487]$ and shown as a translucent orange rectangle. A uniform distribution over the region [1.5, 3.5] was used as a prior, and the value calculated from the frequentism perspective was used to seed the algorithm.



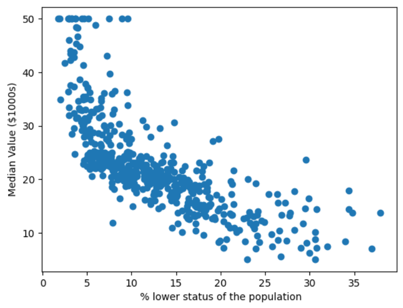
To get the likelihood to be non-zero due to any rounding errors, I had to multiply it by a large positive constant. This is justified because we are interested in the *transition probability*, not the absolute probabilities. In other words, the MH algorithm cancelled out this large positive constant.

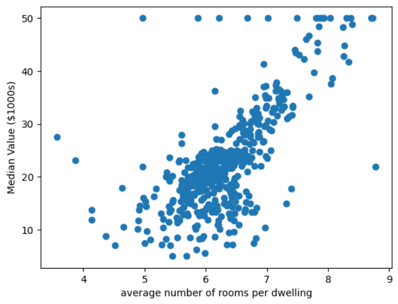
We can use a posterior predictive check to ensure that we've actually found a good value for τ. As we can see from the images, all of the curves (with lifetimes drawn randomly from the random walk) fall extremely close to what we calculated using frequentist methods. It can be difficult to make them out, so I've zoomed in on one of the images:

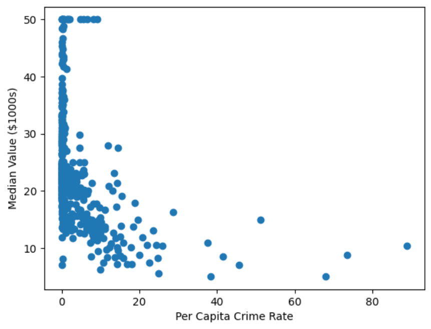


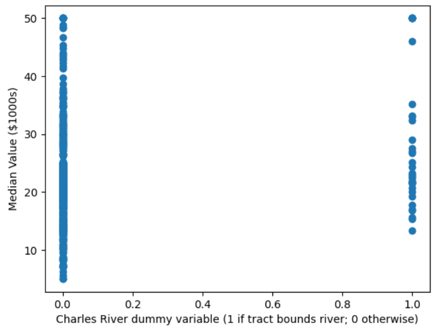
Problem 4

Part A:





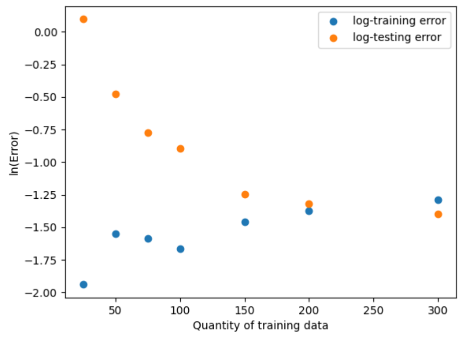




These are four of the graphs produced (others are in the attached folders). From these plots, I think that crime rate, percentage lower status of the population, and the average number of rooms per dwelling will have the largest impact on the median value of the homes. The Charles River also seems to play a moderate role in affecting housing prices.

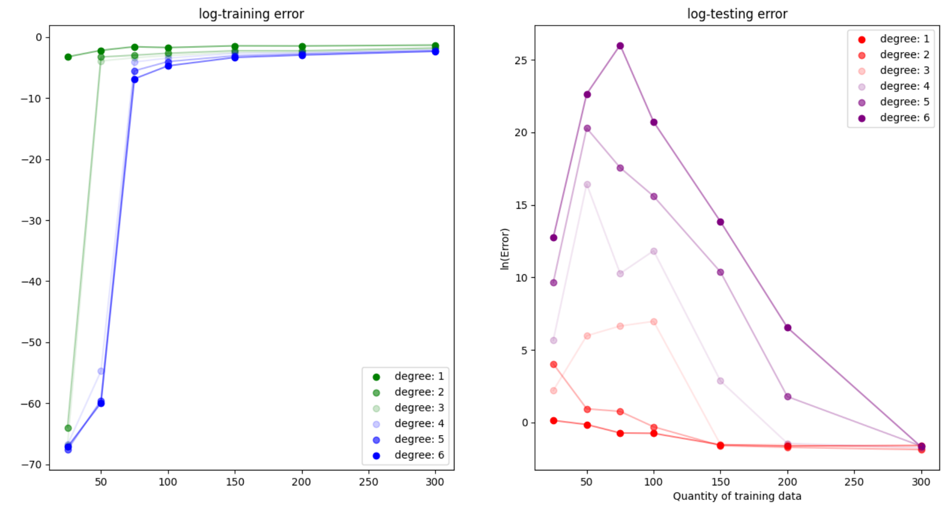
Part B:

The errors were very small, especially for the training (they got flattened onto the x-axis) so I decided to plot a logarithm for most the data. Below is a display showing a typical training-testing error graph depending on how many datapoints were used for training. Training error will tend to increase with increasing data, for a model of a given complexity. Testing error will tend to decrease, because the model gets better and better at predicting new pieces of data. Eventually, the model will approach an "equilibrium" where training and testing error are approximately equal, as shown (approximately) in the graph.

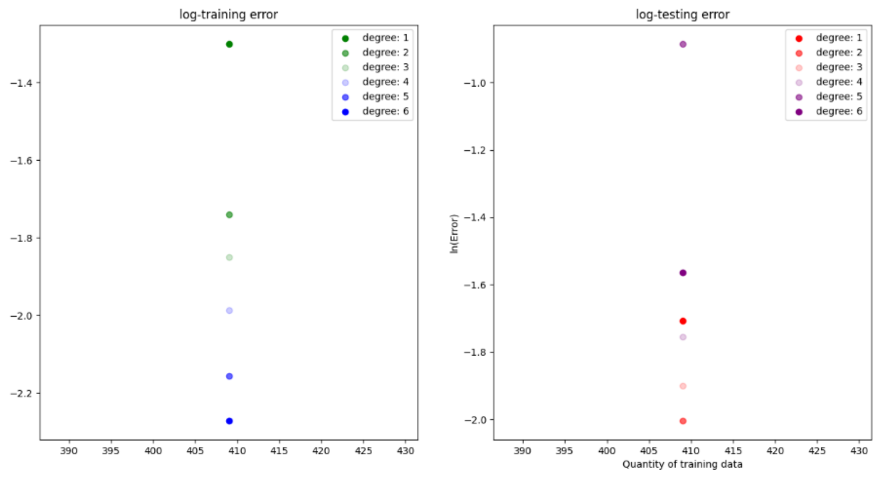


My predictions for what affected housing prices were not always correct. The number of rooms and crime rate were important (as quantized by asking the program to print out the weights) but I underestimated the effects of the Charles River and also of the lower status of the population. I probably should have seen that coming, but it's cool that we can draw intuitions from this!

Part C:



The figure shows the log training and testing error for models of different complexity, and for different data sizes. You can see, if you look carefully, that training error seems to decrease with increasing complexity for all quantities of training data. Testing error is a bit more complex, and is shown better in the next figure. We see that initially the testing error decreases, but then it begins to rise again. This is due to overfitting and the ideal model complexity.

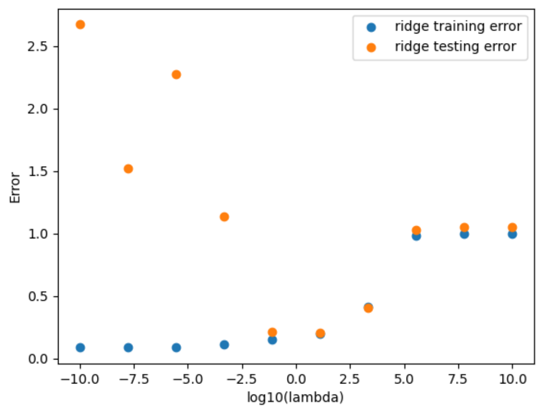


Training error will always go down, because we give the model more degrees of freedom to fit extraneous bits of data. However, this doesn't make it a better predictor. After a certain amount of complexity, the model will actually *decrease* in efficacy, because it's overfitting for the data and making itself useless at predicting new data points.

I was dealing with a *lot* of singular matrices when I increased the complexity of the dataset, to the point where I could never fit a cubic polynomial model. I fixed this by importing and using LinearRegression from sklearn, which dealt with these singular matrices in a much more efficient way than I ever could (I actually am not sure how I would deal with them. Perhaps I'll learn in a more advanced linear algebra class).

Part D:

A way to correct for overfitting is with ridge regression. The results of ridge regression for a sixth degree polynomial are shown in the image, with a wide range of lambda values, so large that it's convenient to plot the log scale.



The training error increases with increasing lambda. This makes sense since we're asking it to fit lots of data on an increasingly restricted (less-complex) model that doesn't have as many degrees of freedom; it's the opposite effect of increasing the order of the polynomial. As usual, the testing error is more complex. We see that it decreases in lambda but then increases. There is an optimal value of lambda for which the model complexity is just right and over-fitting is reduced. However, when the complexity is reduced too far, the model can't account for all the features in the data, and as a result, the training error once again begins to increase.

Problem 5

For this problem I was going to try and be really ambitious and construct a Chinese character classifier based on some data that was released by a national research institution in China. Unfortunately, I couldn't figure out how to read the data, because it wasn't in an easy format to check. Two datasets that I found instead are the records of the people who survived the Titanic, and a spam email set, links here:

https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html

https://archive.ics.uci.edu/ml/datasets/Spambase

I ran Naïve Bayes and Logistic Regression on both databases to predict who would live, in the case of the Titanic, or which emails were spam, in the case of the spam data set.

The Titanic dataset had six features: Passenger class (1, 2, or 3), Sex (Male or Female), Age, number of siblings/spouses aboard, parents/children aboard, and the price they paid for their ticket. The dataset also included names, but I cut these out of the dataset. In total, there were 887 datapoints, and the outcome of each passenger's journey was encoded as an integer.

The spam email dataset had 57 features. 48 of these were word frequencies as they appeared in the email (e.g. conference, money, free), 5 of them were punctuation frequencies, and the final three were related to capital letters. Specifically, they measured the average length of strings of consecutive capital letters, the longest continuous string of captial letters, and the total number of consecutive capital letters. Whether the email was spam or not was encoded as an integer.

Spam Emails

Naïve Bayes had a score of 0.8165, which is not bad for predicting which emails you'd want to ignore in your inbox. Logistic Regression did better with a score of 0.934. This makes sense for this particular dataset, because the connections between words is very important, and I think that completely ignoring any interactions in their joint likelihoods really does scale down the amount of insight you get into the email and its intentions. A variety of banks and technicians could email you talking about finances or tech support without raising any suspicions, but hearing that I need to transfer funds to fix some issue on my laptop immediately raises red flags. I'm guessing that Naïve Baye's can't do this analysis by its very nature.

Titanic Dataset – Predicting Survival

Naïve Bayes had a score of 0.7697, while logistic regression had a score of 0.8025. Compared to the spam emails, this is a much less pronounced difference. The dataset is much smaller, so the lower accuracy rate is understandable. I imagine that the joint-probabilities of different passenger attributes is much lower than the correlation between words in an email, so Naïve Baye is much closer to reality in this dataset, manifesting itself as a score which is not significantly lower than logistic regression.

Titanic Dataset – Predicting Survival

To test multi-class classification problems, I reversed some of the data in the titanic dataset, and asked the algorithm to predict the passenger class of each person (either 1, 2, or 3) based on their ticket price, survival outcome, gender, and other factors. Naïve Bayes had a score of 0.736, while softmax regression (automatically included in sklearn's logistic regression methods) had a score of 0.8222.

How the algorithms are implemented

Naïve Bayes, as I've already alluded to multiple times, asserts that the feature variables are independent of each other. This reduces the complexity of the likelihood quite significantly, as we can write it as a produce of likelihoods of individual variables given the parameter $P(x_i, y)$.[3] I tried to read through the source code on how this was implemented in the sklearn package, but I quickly got lost in a string of helper functions and recursive methods. However, I believe that they somehow implemented a gradient ascent algorithm to find the maximum value of the posterior.

Because of a timeout error on logistic regression, I noticed that the algorithm I imported from sklearn uses the Limited (Memory) Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS), which implements gradient descent. While the details elude me, what I have gathered from my wikipedia reading is that it encodes the curvature into the choice for the next step via the Hessian, while also incrementally improving its estimations of the Hessian. I assume that they create either a logistic or softmax function, and employ the L-BFGS algorithm to find the minimum.

[3] https://scikit-learn.org/stable/modules/naive_bayes.html

Ethics question

ML practitioners should be very aware of the limitations of their models, and also very aware of what data they are training on. This is just good science, but I think that sometimes the Artificial Intelligence and Machine Learning tags can make people feel like any work that they do will end up perfectly crafted just because it's done by a computer. The general public is frequently caught up with flashy ML demonstrations and isn't aware of some of its shortcomings; since it's unrealistic for everyone to know in-depth computer science, ML practitioners have an even larger duty in presenting pitfalls of their algorithms.

People who are building these models should also be considerate of where their data is coming from, how it was collected, and what biases or omissions it might have. It's becoming obvious that these algorithms have biased results depending on what group they are classifying (for instance, in facial recognition of Caucasians vs Black People), and we've got to examine our data carefully to ensure that we aren't feeding junk into our machines. It's an unfortunate reality that minorities will have less available data; this is hard to mitigate and I'm not exactly sure how you would or could do it, but it also seems that some computer scientists completely disregard the question.

Data can also be collected very unethically, and I don't think we should be complicit with those unethical practices. For instance, China was building a complete database of Uighur People's DNA with the help of top companies and scientists in the US, likely coercing people to give blood. There's little doubt in my mind that this data is being employed to help China tighten its grasp on its western provinces.[4]

In addition to being aware of where the data is coming from, programmers should also think about how their work will influence society as we need to remember that nothing is ever done in a vacuum, and computer scientists can wield enormous power for good and evil. Some software has obvious societal impacts, such as programs which dictate police presence in different communities. The impact of others is more nuanced and harder to follow, but still very important. For instance, what has been the effect of automating FICO scores or other financial records? What about job selection, stock market trading, or job selection?

Something that stuck with me from our conversation on Friday was how interdisciplinary approaches will most likely be necessary to figure out answers to some of these questions. Computer Science, the Internet, and internet society comprise a huge domain of human experiences, and it's unlikely that any individual with a limited skillset will be able to completely resolve the issues of machine learning. In addition to computer scientists, we need lawyers, philosophers, psychologists, economists, and policy-makers to approach our issues of ethics from multiple perspectives, just because it is such a huge issue.

[4] https://www.nytimes.com/2019/02/21/business/china-xinjiang-uighur-dna-thermo-fisher.html