PREDICTING CANCER DEATHS

A Technical Paper submitted to the Department of Engineering and Society
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By

Turner Hamlin

April 26, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR
Haifeng Xu, Department of Computer Science

Predicting Cancer Deaths

A Machine Learning Approach

Jefferson Hamlin
University of Virginia
jth4xt@virginia.edu

ABSTRACT

This project aims to use some machine learning algorithms to predict a person's chances of dying from lung cancer in the state of Virginia. In addition to predicting mortality rates from the disease, other statistics regarding having cancer, e.g., annual mortality rate, overall mortality rate, cancer patient demographics, income association, etc. will also play a role in my analysis. The purpose of this project is to highlight the most influential factors in determining the rate of dying from cancer in Virginia for the benefit of efficient funding allocation for treatment and general public awareness.

1 INTRODUCTION

For my technical Project I wanted to work on something that dealt with a real issue society faces. For this reason, I decided to focus on cancer rates in the state of Virginia. Cancer is the 2 highest leading cause of death in the United States. Given such a high death count, I wanted to try and isolate some of the key factors that contribute to this in order to have a better understanding of where resources could be better allocated. According to the National Cancer Institute [1], funding predominately goes into understanding, preventing, detecting, diagnosing, and treating cancer as seen in Figure 1. While all of these are very important things, that leaves only a relatively small portion of the budget to be allocated for predicting cancer.

One goal of this project then is to determine whether such a budget could be better adjusted to include more funding for predicting cancer in individuals which could potentially give insights into the other programs. One such potential is if we can better understand what demographic factors make one more likely to develop cancer, it could increase our effectiveness at preventing it in the first place.

Another goal of this project is to measure the effectiveness of different algorithms on the problem. By studying multiple regression models, I can gain better insight into how the factors correlate to one another in their affects on one's chances of developing and/or dying from cancer.

Having taken both machine learning and Data analysis, I had a keen interest in the algorithms these classes implemented so I wanted to incorporate them in to my project

Figure 1: National Cancer Institute proposed budget increase for 2021

2 BACKGROUND

My data was compiled by Noah Rippner [2] and posted on data.world. It consists of a CSV file that contains many parameters he felt were relevant due to their potential impact on lung cancer instances in Virginia.

3 SYSTEM DESIGN

3.1 Preparing the Data.

Rippner's dataset had a lot of useful information and statistics for me to base my model off of, but it also contained many data points that would be extraneous information or were not independent variables.

3.1.1 Visualizing the Data. I first wanted to get a better understanding of what factors were the most important and to visualize the data. For this, I checked to see if any of the categories were missing any data points and found that some of the columns were not completely full, so I knew I would have to account for this later. I then plotted out some the data in histograms to get a better view of what values were for

each category. Next, I wanted to understand how each category affected one another so I used scatter plots to depict the relationships certain values had on one another.

3.1.2 Removing Dependent or Extraneous Data. Having a better understanding of my data, it was now time to modify it to suit my needs. The first step was to remove any unwanted categories that either contained extraneous information or was dependent on a different column of data. For example, the data set contained values for median age, median age of females, and median age of males. These 3 values were clearly dependent on each other and would potentially skew my prediction should I include all of them. For that reason, I omitted such cases in order to only use what I determined to be necessary data. The set also included statistics for education level and financial status. I decided that one's wealth was, at least partially, dependent on one's education level, and so I omitted the latter since financial status seemed like it was more likely to affect one's chances of surviving lung cancer anyway.

3.1.3 Replacing Null Data Points. Having removed the extra data and being left with only the essential data, I still had to adjust for any value that was not recorded and would in turn present problems when trying to perform any prediction on it. In order to fix this, I first calculated the mean value for each column and subtracted it from every data point in that respective column. Having done so, the mean for each column would now be 0. I then divided every value by the standard deviation in order to standardize my data so that every value was a given a Z score, or a measurement of how many standard deviations that value is away from the mean.

I was then able to account for all of the null data by assigning each missing point a value of 0. This was essentially just assigning them the value of the average for that category since given the new mean was now 0. This ensured that it didn't significantly alter my data while still allowing me to use the entries that were missing some of the statistics. I then printed out the non-null counts for every category in order to ensure they were all the same size which would mean I no longer had any missing data.

3.2 Preparing the Regression Models.

Having adjusted my data to a point where I could begin to run a model, I had to choose an algorithm that would best suit my needs. Since machine learning has many different algorithms where no single algorithm works the best for any problem, I needed to try a couple of different approaches to try and find one that most accurately predicts the death rate according to the data. Since I already had results for the desired output (TARGET_deathrate), I knew I would want to use a supervised learning algorithm. This type of model uses training data to fit the data to come up with a prediction and then uses test data to check how accurate the results are.

3.2.1 Multiple Linear Regression. For my first algorithm, I decided to implement multiple linear regression (MLR). Linear regression attempts to fit a linear model to a set of data based on a single input parameter in order to be able to predict an output given by such a line. Multiple linear regression uses multiple input parameters in order to develop a linear best fit line to predict a single output. I used this method because linear regression is often regarded as the standard regression model and I felt confident in its implementation. Formula 1 depicted below shows the equation for multiple linear regression.

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_n x_n \quad (1)$$

Here, y represents the given output values that the model is attempting to predict, b_n represents the weight value given to the nth input parameter, and X_n represents the nth input parameter.

3.2.1.1 Gradient Descent. Gradient descent is an optimization method for regression models that attempts to minimize a cost function by iteratively updating its parameters according to a learning rate until it reaches a local minimum, therefore optimizing the learning algorithm being used. Essentially, it takes the given parameters and assigns a weight value to each of them. It then fits the data using those weight values in whatever regression model is being used. After finding a new prediction and evaluating its cost, it updates the weight values for the parameters by moving down the gradient according to a pre-defined learning rate until it reaches a minimal value for the cost function.

3.2.1.1.1 Initial Setup. Since linear regression is a supervised learning algorithm, I first needed to split my data into an X and a Y set which represented the parameters that affect the outcome and the outcome itself, respectively. From there I had to split the groups into a training set and a test set. I did this randomly to ensure there was no bias but made sure that the train values for X corresponded with their correct Y values and did the same thing for my test data.

I then established my parameter vector called theta. This vector is initialized with random values to start, and then iteratively updated. It contains the weights for each parameter and each time gradient descent is run, it reevaluates these weights to try and find the optimal solution that gives the best predictions based on the training data. Afterwards, I initialized my learning rate, alpha. This value determines how much of a step to take down the gradient. If it is too small, the algorithm will take way too long to reach

the minimum. If it is too large, the algorithm will go farther past the minimum than our previous point and will never converge as it will just continue to get farther and farther away from the minimum value. I started off with a very small value for this to ensure it would converge and later adjusted it in in accordance to my cost function.

3.2.1.1.2 Implementation. Once I had my initial values set, I could begin the actual prediction through linear regression with the equation defined below in formula 2 for gradient descent.

$$\theta = \theta - \eta \frac{2}{m} \mathbf{X}^T (\mathbf{X}\theta - \mathbf{y}) \tag{2}$$

Here, θ is the parameter vector as described earlier which contains the weights to be used for each one of the input values. η is the learning rate I established to be alpha. M is the number of train vectors used so I set this value to the length of the X train vector. X is the data used to make the prediction so in this case it would be X train. The transpose is used here in order to keep the dimensions correct for the multiplication. Y corresponds to the output vector used to make the predictions as well so I used Y train. Once I had implemented my formula, it was just a matter of plugging in the variables.

First, I took the dot product of my X train data with my parameter vector, theta. I then plug in this new array into the equation and subtract the y train values from it. After taking the dot product of the Transpose of X train with the difference I had just found, I multiplied it by 2 and divided by m. This gave me a set of gradients for each parameter to be used to calculate the new weights for each parameter in theta. I did this by multiplying the gradients vector with my learning rate, alpha, and then subtracting this resulting vector from theta to create a new theta vector. With this new parameter weights vector, I used linear regression to make a prediction based on the X test data by taking the dot product of the X test values with my new theta vector.

With these predictions, I compared them to my actual results given by Y test and found the root mean square error (RMSE). I did this by using the mean_square_error function that is built into the sklearn.metrics class. This function calculates the mean square error by taking the distance of each prediction to its actual value, squaring this value, and then taking the average of all of these squared errors. Once I had this measurement, I took the square root to give me the root mean square error and stored this value in an array to compare to my other RMSE values to ensure that it was gradually decreasing with each iteration.

Once I had this set up, I could then repeat the steps for gradient descent to calculate a new weights vector for my parameters, form a new prediction, calculate a new root mean square error, and compare these errors to ensure that the RMSE decreased gradually with more iterations until eventually leveling out at a certain value. At this point, the benefit of more iterations decreases exponentially and there is little reason to continue readjusting the parameter vector.

However, the next value that did still need adjusting was my learning rate, alpha. As I mentioned earlier, this value determines the step size, or how far down the gradient the value will move. I started this value off very small to ensure that it would at least converge even though it might take quite some time and would unlikely be optimal. In order to do this, I tested many alphas to gradually determine one that would return the best root mean square error for my predictions. After many tests, I settled on a value of 0.01 as this resulted in the best RMSE.

3.2.1.2 Normal Equation. In order to ensure this was optimal, I wanted to compare my gradient descent result to another optimization method that also determines a weight vector that will yield a minimal root mean square error. In order to do so, I implemented the Normal Equation. This optimization method is great for small data sets as It doesn't require many iterations to perform and can be implemented very quicky. However, since it uses the inverse of a matrix in its equation, it does not scale well and will be extremely slow for large data sets. Given that the data set I am working with is reasonably small, this was not a concern. Formula 3 depicted below is the equation used for such an optimization method.

$$\theta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \tag{3}$$

Again, θ represents the weights vector for the parameters, X represents the input training set, and y represents the output training set. Some advantages of this method are that it does not require the use of a learning rate (alpha), the number of training set entries (m), and it does not require many iterations in order to find the optimal solution. For a small data set like the one I used, this a great method for optimizing the weights of my parameters. I took the transpose of my training input, X train, and multiplied that by X train. I then calculated the inverse of this new square matrix, multiplied it by the transpose of X train, and finally multiplied it with Y train. This gave me my optimized weights for my parameters that should minimize the cost function.

3.2.2 Multiple Polynomial Regression. After completing a linear regression model with 2 different optimization methods, I wanted to test other models to see if a different approach might be more accurate. I decided to implement a multiple polynomial regression (MPR) using sklearn's polynomial features attribute. This allowed me to input my

X train set so that it could append the necessary polynomial features into my training set.

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n \quad (4)$$

$$Y = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + b_3 \cdot x_1 \cdot x_2 + b_4 \cdot (x_1)^2 + b_5 \cdot (x_2)^2 \quad (5)$$

Equation 4 provided by Subarna Lamsal [3] depicts the formula for polynomial linear regression which uses only a single input parameter but can be raised to the nth power in order to better fit certain problems. However, given that this problem uses many inputs, this equation needed to be modified in order to be a multiple polynomial regression equation so that it could include both multiple parameters and multiple degrees of power. This new modified equation is depicted in formula 5 which defines a multiple polynomial regression for a power of 2. It's easy to then create the equations for higher powers following this same pattern for each extra parameter and sequential degrees of power.

Essentially, it creates polynomial parameters for each input category and then plugs them into the multiple linear regression equation such that X_2 in MLR is equal to $(X_1)^2$ in polynomial linear regression and so on. Using this formula, I fit my new data using sklearn's linear regression with both a power of 2 and 3.

3.2.3 Decision Tree Regression. I wanted to build one more model to ensure that I had covered a wide arrange of regressions to be able to feel confident that one of them would produce an acceptable set of predictions. For the final model, I decided to implement a decision tree regression to predict the death rate for cancer. Using the same data for my train and test sets to ensure there was no bias, I fit my training data to sklearn's Decision Tree Regressor. This type of regression involves creating a tree to sort the data by dividing it into smaller and smaller subclasses. It creates these splits by minimizing the residual sum of squares (RSS) between the actual value and the average for that node. It goes through the data at a certain node and tests a certain split and checks the RSS for that split. It stores this value and then checks the next potential split. After checking all options, the least RSS value becomes the new split for the tree.

By continuing this process, it attempts to split the data into the smallest categories it is able to. Once it has done this, it can take an input and follow the tree that most similarly correlates to its input values to find its predicted result for the output. The decision tree identifies the factors that have the most importance and splits them first and continues through the data to the least important parameters until the tree is expanded as far is it can be. At this point, I used my X test data set to get predictions from the decision tree and then compared those predictions to my Y test set to find the RMSE.

4 RESULTS

4.1 Multiple Linear Regression

4.1.1 Optimized with Gradient Descent. After implementing gradient descent in order to optimize my weight parameters, I could test my model to find its root mean square error value to determine how close it was to predicting the actual values of the test set. Using an alpha value of 0.01 and a randomly initialized theta vector, I produced an RMSE value of 0.747. I also compared this error to those produced by the same method but with different alpha values. Figure 2 shows the decreasing of the error values as more iterations occur for 6 different alpha values.

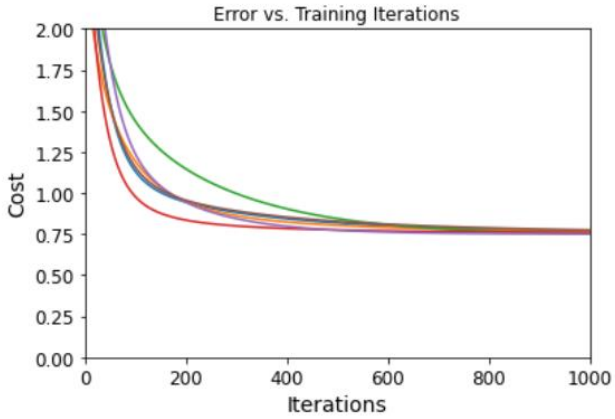

Figure 2: RMSE vs iterations for alpha values of [0.0065, 0.007, 0.0075, 0.008, 0.01, 0.015]

The purple curve is the line created by the regression model with an alpha value of 0.01. While its error starts higher than some of the other alphas and decreasing a slower, with more iterations it eventually becomes the most effective value with smallest RMSE value (cost). Given the drastic decrease in cost for the first 200 iterations or so, it is clear that the implementation of gradient descent was certainly worthwhile.

4.1.2 Optimized with Normal Equation. With the described execution of the normal equation to optimize a multiple linear regressor, the resulting RMSE value came out to 0.750. While very close, this is slightly greater than the gradient descent optimization method so I felt confident that gradient descent was the better method for this particular problem.

4.2 Multiple Polynomial Regression

4.2.1 2nd Degree Polynomial. For my first implementation of multiple polynomial regression, I used a second-degree polynomial. This model produced an RMSE value of 0.823 for its predictions of the X test data set. While quite accurate, it was not any better than the MLR model.

4.2.2 3rd Degree Polynomial. My second implementation of MPR used a third-degree polynomial to see if adding another power would increase the accuracy of the regression. However, with an RMSE value 29.223, it was clear that this was not the case and this model was not a useful regression for this specific problem.

4.3 Decision Tree Regression

My final regression model was the decision tree regression. This model was much more accurate than the 3rd degree MPR, but failed to produce an RMSE value less than either of the 2 linear regression model approaches. With a final error of 1.042, its safe to regard this model as ineffective for this specific problem.

CONCLUSION

While the first implementation of multiple linear regression with gradient descent for optimization ended up being the most accurate prediction model, that does not mean that the implementation of the other regression models was not worth the effort. Knowing that these other methods did not perform as well gives me much greater confidence in the first model. Without actually testing other regressions, I would have no way of knowing whether these implementations could potentially produce more accurate predictions.

With the growing capabilities of computers, machine learning algorithms are continuing to evolve to make predictions and classifications more accurate and in less time. These models have many implications that can help us isolate certain parameters in problems to give us a better understanding on how to fix them. For example, my model showed that the percent of Asians in a given population positively correlates to a higher chance of death. This type of information gives us a greater insight into where we should focus our efforts. Knowing that Asians are at a higher risk, as a society we should put extra emphasis into studying the reasons behind this. This project gave me a much better understanding of some of the factors that put certain individuals at a greater risk and could potentially be used to help mitigate those risks in the future

5 FUTURE WORK

Having tested a wide range of regression models, I feel confident that the multiple linear regression model with gradient descent optimization is a strong candidate for making predictions on this dataset. However, that's not to say that it cannot be made more accurate.

If I were to continue working on this project, I would be interested in implementing some more parameters that could potentially have a great effect on one's chances of surviving lung cancer. For example, implementing a parameter representing an area's proximity to a hospital with a cancer specialist could potentially increase the accuracy of this model as it would be logical to conclude that the closer one is to the medical help they need, the more likely they are to survive any complications.

REFERENCES

[1] NCI. Annual Plan & Budget Proposal for Fiscal Year 2021. https://www.cancer.gov/about-nci/budget/plan/2021-annual-plan-budget-proposal.pdf

[2] Noah Rippner. 2017. OLS Regression Challenge. (January 2017). Retrieved February 20, 2021 from https://data.world/nrippner/ols-regression-challenge

[3] Subarna Lamsal. 2021. Multiple Linear Regression: Sklearn and Statsmodels. (February 2021). Retrieved March 13, 2021 from https://codeburst.io/multiple-linear-regression-sklearn-and-statsmodels-798750747755