CPSC 532M Assignment 3

Lucas Palazzi, Student #79285755
Gustaf Backman, Student #17025362

1 Finding Similar Items

1.1 Exploratory data analysis

1.1.1 Most popular item

The item with the most stars is Item B000HCLLMM, a Grill Cover. It has 14454 stars.

1.1.2 User with most reviews

The user who has rated the most item is user A100WO06OQR8BQ. This user has rated 161 items.

1.1.3 Histograms

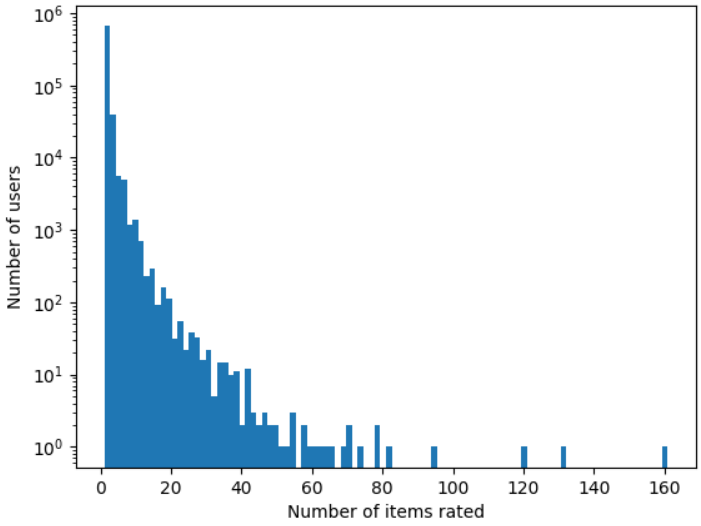

Figure 1: Number of ratings per user.

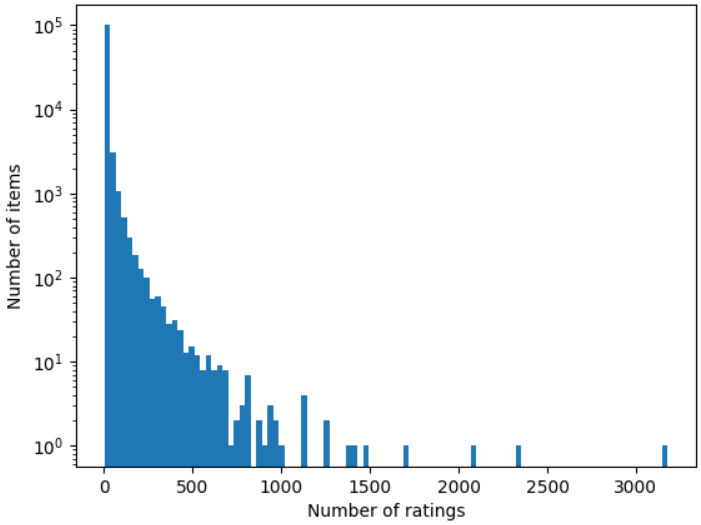

Figure 2: Number of ratings per item.

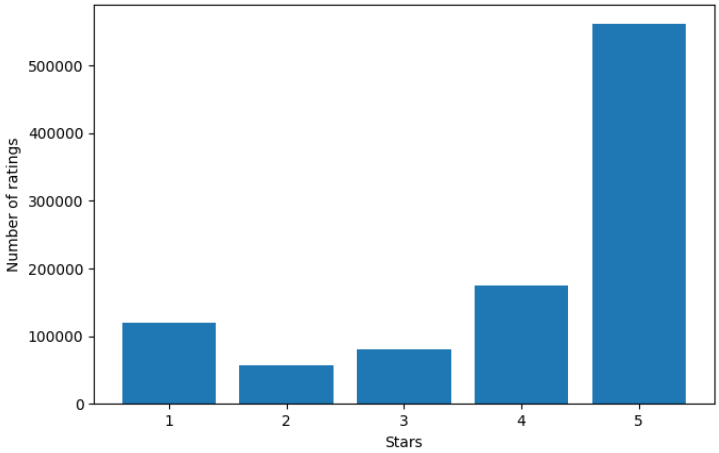

Figure 3: Number of ratings per rating.

1.2 Finding similar items with nearest neighbours

1.2.1 Euclidean distance

URL for the nearest items:
https://www.amazon.com/dp/B00IJB5MCS
https://www.amazon.com/dp/B00IJB4MLA
https://www.amazon.com/dp/B00EXE4O42
https://www.amazon.com/dp/B00743MZCM
https://www.amazon.com/dp/B00HVXQY9A

1.2.2 Normalized Euclidean distance

URL for the nearest items:
https://www.amazon.com/dp/B00IJB8F3G
https://www.amazon.com/dp/B00IJB5MCS
https://www.amazon.com/dp/B00IJB4MLA
https://www.amazon.com/dp/B00EF45AHU
https://www.amazon.com/dp/B00EF3YF0Y

1.2.3 Cosine similarity

URL for the nearest items:
https://www.amazon.com/dp/B00IJB5MCS
https://www.amazon.com/dp/B00IJB8F3G
https://www.amazon.com/dp/B00IJB4MLA
https://www.amazon.com/dp/B00EF45AHU
https://www.amazon.com/dp/B00EF3YF0Y

1.3 Total popularity

With the euclidean metric, the number of reviews were:

$$[55, 45, 1, 1, 1]$$

Whereas the cosine metric chose predictions with these reviews:

$$[55, 91, 45, 66, 110]$$

This makes sense according to the concept of normalizing the distance. The 3 last predictions with Euclidean metric (with only 1 review each) probably coincide very well with the input, but does not take popularity into account. The found neighbors by cosine similarity do however have more reviews which is in line with the theory.

2 Matrix Notation and Minimizing Quadratics

2.1 Converting to Matrix/Vector/Norm Notation

1. $\|Xw - y\|_\infty$
2. $\hat{y}^\intercal V\hat{y} - 2\hat{y}^\intercal y + y^\intercal Vy + \frac{\lambda}{2}w^\intercal w$
3. $\|Xw - y\|_1^2 + \frac{1}{2}\|\Lambda w\|_1$

2.2 Minimizing Quadratic Functions as Linear Systems

Work is shown, final answers are enclosed in boxes.

1.

$$f(w) = \frac{1}{2}\|w - v\|^2 = \frac{1}{2}\sum_{j=1}^{d}(w_j - v_j)^2$$

$$\frac{\partial f(w)}{\partial w_j} = \frac{1}{2}(2)(w_j - v_j) = w_j - v_j$$

$$\therefore \nabla f(w) = w - v = 0 \implies \boxed{w = v}$$

2.

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^\intercal \Lambda w = f_1(w) + f_2(w)$$

$$f_1(w) = \frac{1}{2}\|Xw - y\|^2 = \frac{1}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} x_{ij}w_j - y_i\right)^2$$

$$\begin{aligned}\frac{\partial f_1(w)}{\partial w_j} &= \frac{1}{2}\sum_{i=1}^{n}(2x_{ij})\left(\sum_{j=1}^{d} x_{ij}w_j - y_i\right) \\ &= \sum_{i=1}^{n} x_{ij}\left((Xw)_i - y_i\right) \\ &= \sum_{i=1}^{n} x_{ij}^\intercal (Xw)_i - \sum_{i=1}^{n} x_{ij}y_i \\ &= (X^\intercal Xw)_j - (X^\intercal y)_j \implies \nabla f_1(w) = X^\intercal Xw - X^\intercal y\end{aligned}$$

$$f_2(w) = \frac{1}{2}w^\intercal \Lambda w \implies \nabla f_2(w) = \Lambda w \text{ (since } \Lambda \text{ is symmetrical)}$$

$$\therefore \nabla f(w) = \nabla f_1(w) + \nabla f_2(w) = X^\intercal Xw - X^\intercal y + \Lambda w = \left(X^\intercal X + \Lambda\right)w - X^\intercal y = 0$$
$$\implies \boxed{\left(X^\intercal X + \Lambda\right)w = X^\intercal y}$$

3.

$$\begin{aligned}f(w) &= \frac{1}{2}\sum_{i=1}^{n} v_i(w^\intercal x_i - y_i)^2 + \frac{\lambda}{2}\|w - w^0\|^2 \\ &= \frac{1}{2}\sum_{i=1}^{n} v_i\left(\sum_{j=1}^{d} w_j x_{ij} - y_i\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\left(w_j - w_j^0\right)^2\end{aligned}$$

$$\begin{aligned}\frac{\partial f(w)}{\partial w_j} &= \frac{1}{2}\sum_{i=1}^{n}(v_i)\left(2x_{ij}\sum_{j=1}^{d} w_j x_{ij} - y_i\right) + \frac{\lambda}{2}(2)\left(w_j - w_j^0\right) \\ &= \sum_{i=1}^{n} x_{ij}v_i(w^\intercal x_i) - \sum_{i=1}^{n} x_{ij}v_i y_i + \lambda\left(w_j - w_j^0\right) \\ &= \left(X^\intercal VXw\right)_j - \left(X^\intercal Vy\right)_j + \lambda\left(w_j - w_j^0\right)\end{aligned}$$

$$\therefore \nabla f(w) = X^\intercal VXw - X^\intercal Vy + \lambda\left(w - w^0\right) = \left(X^\intercal VX + \lambda I\right)w - X^\intercal Vy - \lambda w^0 = 0$$
$$\implies \boxed{\left(X^\intercal VX + \lambda I\right)w = X^\intercal Vy + \lambda w^0}$$

3 Robust Regression and Gradient Descent

3.1 Weighted Least Squares in One Dimension

See README.md for link to code.

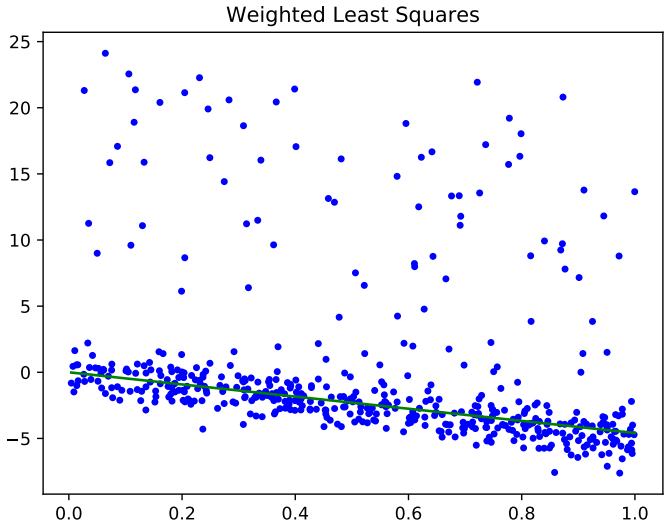

Figure 4: Plot obtained by weighting the data points.

3.2 Smooth Approximation to the L1-Norm

$$f(w) = \sum_{i=1}^{n} \log(\exp(w^\intercal x_i - y_i) + \exp(y_i - w^\intercal x_i))$$

$$g(w) = \exp(w^\intercal x_i - y_i) + \exp(y_i - w^\intercal x_i) \implies f(w) = \sum_{i=1}^{n} \log(g(w))$$

$$\frac{\partial f}{\partial w} = \sum_{i=1}^{n} \frac{g'(w)}{g(w)}, \text{by the chain rule.}$$

$$g'(w) = x_i \exp(w^\intercal x_i - y_i) - x_i \exp(y_i - w^\intercal x_i)$$

$$\implies \frac{\partial f}{\partial w} = \sum_{i=1}^{n} x_i \frac{\exp(w^\intercal x_i - y_i) - \exp(y_i - w^\intercal x_i)}{\exp(w^\intercal x_i - y_i) + \exp(y_i - w^\intercal x_i)}$$

This gives the gradient for w:

$$\nabla f(w) = \begin{bmatrix} \sum_{i=1}^{n} x_{i1} \frac{\exp(w^\intercal x_{i1} - y_i) - \exp(y_i - w^\intercal x_{i1})}{\exp(w^\intercal x_{i1} - y_i) + \exp(y_i - w^\intercal x_{i1})} \\ \sum_{i=1}^{n} x_{i2} \frac{\exp(w^\intercal x_{i2} - y_i) - \exp(y_i - w^\intercal x_{i2})}{\exp(w^\intercal x_{i2} - y_i) + \exp(y_i - w^\intercal x_{i2})} \\ \vdots \\ \sum_{i=1}^{n} x_{id} \frac{\exp(w^\intercal x_{id} - y_i) - \exp(y_i - w^\intercal x_{id})}{\exp(w^\intercal x_{id} - y_i) + \exp(y_i - w^\intercal x_{id})} \end{bmatrix}$$

3.3 Robust Regression

See README.md for link to code.

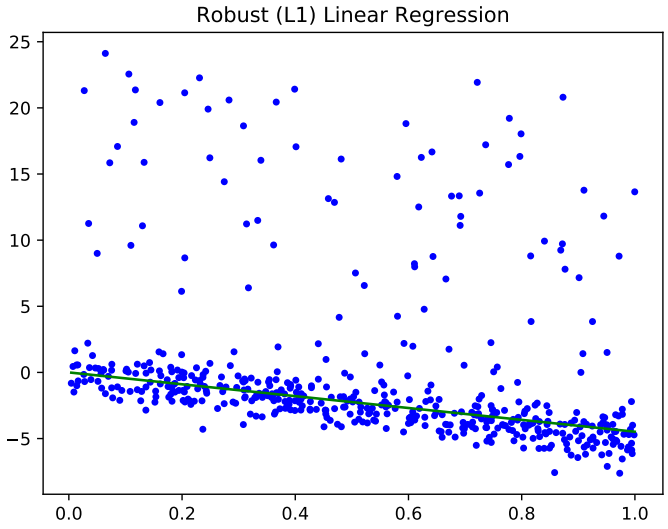

Figure 5: The plot obtained by the smoothing approximation to the absolute value function.

4 Linear Regression and Nonlinear Bases

4.1 Adding a Bias Variable

See `README.md` for link to code.

$$\text{training error} = 3551.3$$

$$\text{test error} = 3393.9$$

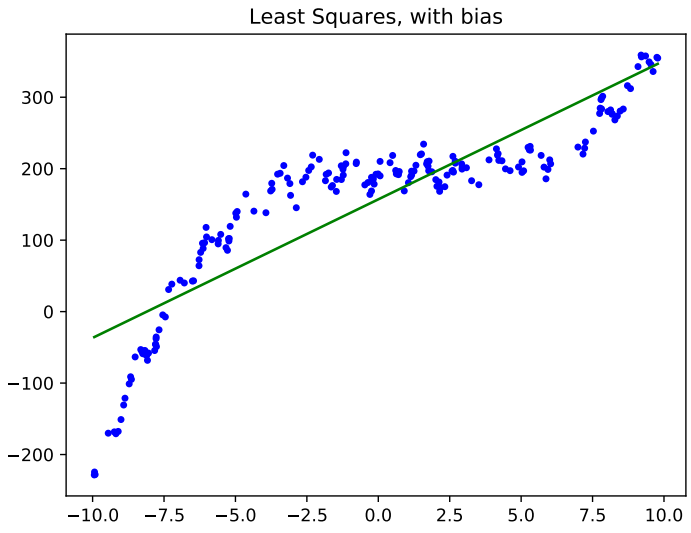

Figure 6: Training data with least squares regression line, with bias (Question 4.1)

4.2 Polynomial Basis

See README.md for link to code.

Below is a table reporting the training and test errors for $p = 0$ through $p = 10$.

p	Training Error	Test Error
0	15480.5	14390.8
1	3551.3	3393.9
2	2168.0	2480.7
3	252.0	242.8
4	251.5	242.1
5	251.1	239.5
6	248.6	246.0
7	247.0	242.9
8	241.3	246.0
9	235.8	259.3
10	235.1	256.3

Effect of p on training error: As p increases, the training error decreases. At first, the training error decreases more dramatically, but at around $p = 3$ the training error begins to decrease at a slower rate.

Effect of p on test error: At first, the test error decreases dramatically as p increases. At around $p = 3$ the error starts to decrease at a slower rate, and at $p = 6$ the error starts to slightly increase or decrease as p is incremented, hovering at approximately 250.

5 Very-Short Answer Questions

1. In k-means, the outlier point will be assigned to the cluster with the nearest mean. In density-based clustering the outlier will not be assigned to a cluster because it is not close enough to a core point (assuming the distance hyperparameter is not too large).
2. Density-based clustering is not as sensitive to initialization as k-means, so restarting the algorithm with another random initialization will not significantly change the clustering.
3. Hierarchical clustering can find non-convex clusters since DBSCAN can be used in hierarchical clustering, and DBSCAN can find non-convex clusters.
4. One way of conducting model based outlier detection is by calculating the Z-score for the data points. If the probability of a data point is below a chosen value, the point will be considered an outlier. A problen here is that the method requires the data to correspond to a normal distribution, which isn't always the case. It also assumes unimodality, which isn't assured at all.
5. An example of graphical-based outlier detection is using a data plot such as scatterplot to visually identify outliers. A problem with this method is that we are limited to looking at only two variables at a time.
6. Supervised outlier detection can be done using decision trees. This can identify complex outliers, but under the condition that training data is available. It would need to know the properties of the outliers in order to find them, which makes it hard to find new sorts of outliers.
7. For linear regression with 1 feature, it would not make sense to use gradient descent instead of normal equations. The cost of using gradient descent is $\mathcal{O}(ndt)$ and the cost of using normal equations is $\mathcal{O}(nd^2 + d^3)$, and since $d = 1$ using gradient descent will give us a higher cost.

8. We typically add a column of ones to X when we do linear regression in order to add an intercept to the linear model. We should not do this with decision trees because if a feature has a value of 1 for all examples it would be irrelevant to the model (you can't make a meaningful decision based on the feature if it is the same for all examples).

9. That a function is convex doesn't imply it has a stationary point, for example $f(x) = \frac{1}{x}, x > 0$ is convex but does not have a stationary point. However, if a convex function has a stationary point, this point is either a global maximum or a global minimum.

10. We must use gradient descent in robust regression because even after smoothing the loss function, there is no closed-form solution to $\nabla f(w) = 0$.

11. Steps that are too small would make the computation too expensive and the descent would converge too slowly.

12. Steps that are too big could miss the stationary point and cause the descent to diverge.

13. The log-sum-exp function is used to do a smooth approximation of the maximum-function. We can then apply gradient descent to the smooth approximation, which we can't apply to the non-smooth maximum-function.

14. Some sort of periodic function, for example sin(x).

6 Project Proposal

Project group members: Lucas Palazzi (f7i1b) & Patrick Boutet (k4i1b)

Project template: Application bake-off

Research area: Error-resilient software

6.1 Background

Fault injection (FI) is the process of systematically introducing errors into a program and observing the outcome. FI is a commonly used tool to test the effectiveness of error detection/correction techniques with respect to *hardware* faults. FI experiments are conducted by running the program-under-test many times (often in the thousands), and each time randomly injecting a fault into the program. As the injection space is very large, typical FI tools use "Monte Carlo simulation" to sample the injection space and obtain a statistical estimate of the outcome probabilities. For a given FI run, the output of the program can typically be classified into three categories:

(1) **crash**, i.e., the program throws an exception and "crashes",

(2) **silent data corruption (SDC)**, i.e., the program output is incorrect, and

(3) **benign output**, i.e., no deviation from the expected output.

Therefore, after completing the desired number of FI runs one can estimate the probability of, say, a crash occurring (given that a fault has occurred) as the number of runs resulting in a crash divided by the total number of runs. The same can be done for SDCs and benign outputs.

As a fairly general term, fault injection can operate at virtually any level of abstraction. For example, we can inject faults directly at the hardware level (e.g., using specially designed hardware components to emulate hardware faults), or we can inject faults somewhere in the software layer (e.g., in the compiled assembly code of the program). The higher the level of abstraction, the easier it is to draw meaningful insights from the results as the findings can be more directly translated to the design of the software. However, raising the level of abstraction comes with a cost in accuracy, as hardware faults occur in the lower levels of the system stack and modeling them at higher levels may be challenging. For this project we are interested in two "levels" of FI:

(1) **high-level FI** operating at the program's intermediate representation (IR) level, and

(2) **low-level FI** operating at the programs assembly code level.

6.2 Problem Description

Low-level FI is considered an accurate method for emulating hardware faults. However, the accuracy of high-level FI is not as conclusive. High-level FI has been found to be accurate for emulating hardware errors that cause SDCs, but not crashes. This is likely caused by the effects of different optimizations in the compiler back end that effect memory operations.

Since high-level FI is easier to work with as it operates closer to the source code level, a method of obtaining a more accurate estimate of the crash probability without the need to run any low-level FI experiments is desirable. Furthermore, FI experiments can take a very long time to run, depending on the size of the program being tested, so it is often not feasible to conduct both high-level and low-level FI experiments. Therefore, the research question we ask in this project is as follows:

Can we predict the crash probability as estimated by low-level FI using the results of high-level FI experiments?

6.3 Overview of Data Set

In order to predict the crash probability as estimated by low-level FI, we consider a training data set with 3 features ($d = 3$). Each data point x_i is a different benchmark program that was tested using both high-level and low-level FI. Each program is also profiled at both the IR and assembly level to determine the percentage of total instructions that operate on memory addresses, called *memory-dependent instructions*. The outputs y are the estimated crash probabilities as measured by low-level FI.

The features for each x_i are as follows:

$$x_{i1} = \text{estimate of crash probability as measured by high-level FI}$$
$$x_{i2} = \text{percentage of memory-dependent instructions in IR code}$$
$$x_{i3} = \text{percentage of memory-dependent instructions in assembly code}$$

The expected size of the training data set is $n = 25$. We plan to also include a test set with a size between $n = 10$ and $n = 20$. The current size of the data set could potentially be larger if we are able to collect more data, however FI experiments can take a very long time to run depending on the program so this is not guaranteed.

6.4 Machine Learning

For this project we will try a variety of regression machine learning techniques in order to determine which model best fits the data. As far as we know, nobody has tried to use machine learning with this type of data. The types of ML models we use will be limited to regression, as the predictions of the model are continuous values. Our goal is to find a model, or a handful of models, that can predict a given program's crash probability with a reasonable degree of confidence.

6.5 Contributions

This project will aim to provide some insight into whether machine learning is a viable tool to use in predicting fault injection experiment results. Furthermore, it will provide some insight into whether using a metric related to the amount of *memory* operations in a program is useful in predicting the crash probability measured by a set of FI experiments.