CP8318/CPS803 Machine Learning

Assignment 1

Rohaan Ahmed
PhD Student - Computer Science
rohaan.ahmed@ryerson.ca

October 4, 2020

Instructor: Dr. Nariman Farsad
October 4, 2020

1. Linear Regression

1.1. Learning degree-3 polynomials of the input

$$J(\theta) := \frac{1}{2}\sum_{i=1}^{n}\left(y^{(i)} - \theta^T\phi(x^{(i)})\right)^2 \tag{0.1}$$

$$\theta \leftarrow \theta + \alpha\sum_{i=1}^{n}\left(y^{(i)} - \theta^T\hat{x}^{(i)}\right)\hat{x}^{(i)} \tag{0.2}$$

where:
$\theta^T\phi(x^{(i)}) = h_\theta\left(x^{(i)}\right)$
$n = 3$
α = the learning rate
$\hat{x}$ is as described in the question

1.2. Coding question: degree-3 polynomial regression

Observations

1. The Normal Equation provides the "best-case" closed form solution to the estimation problem for this dataset.

2. It can be seen from Figure 1 that, with $k = 3$, the model does not capture the distribution of the given dataset extremely well, even with the Normal Equation. Thus, $k = 3$ produces a model that "under-fits" the data]

3. We hypothesize the following based on the results observed in section (1.1):

- As we increase k (i.e., the mappings x^k), we would get better fitting to the dataset.
- As we increase k, the complexity of the algorithm would increase, thus requiring greater computation constraints and finer parameter tuning. At some point, increasing k would become counter-productive, resulting in model "over-fitting" and unreasonable computational constraints.
- Finally, considering that the dataset roughly resembles a sinusoidal function, using sinusoidal mapping would result in "good-fitting" models, even for lower k values.

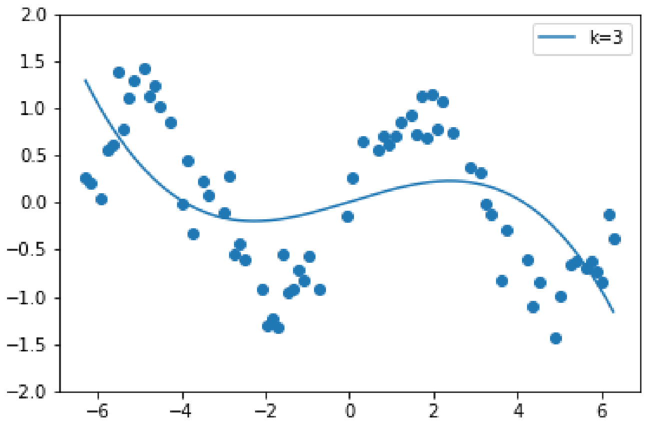

Figure 1: Normal Equation with Polynomial Feature Mapping k = 3

1.3. Coding question: degree-3 polynomial GD and SGD

Observations

1. Figure 2 shows the result of running the Batch Gradient Descent algorithm until convergence is achieved for $k = 3$, with θ initialized randomly. "Convergence" here means when updates to every θ_j is below a certain threshold given by a chosen *Convergence Threshold* (in this case, $1e^{-9}$). The final plot matches the plot obtained using the Normal Equation.

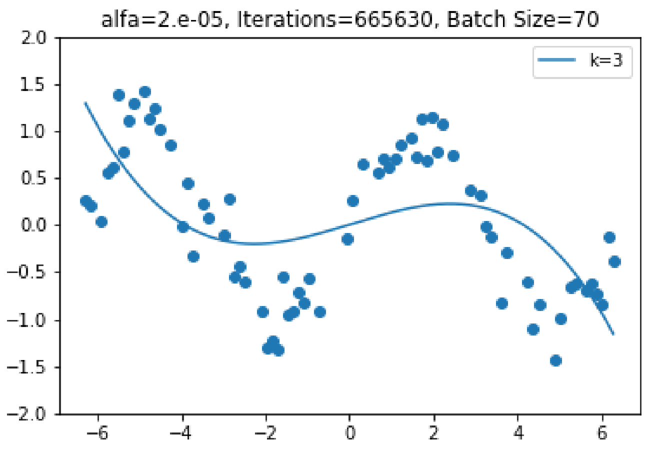

Figure 2: Batch Gradient Descent with Polynomial Feature Mapping k = 3

2. Figure 3 shows the result of running the Stochastic Gradient Descent algorithm until convergence is achieved, with θ initialized randomly. The final plot matches the plot obtained using the Normal Equation.

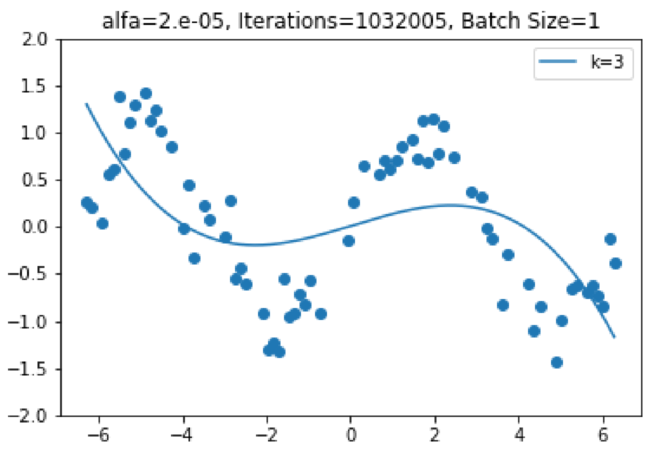

Figure 3: Stochastic Gradient Descent with Polynomial Feature Mapping k = 3

3. When values of θ are randomly initialized, the number of iterations the algorithm takes to converge to steady-state varies, as expected. In general, 100 and 1000 iterations are not sufficiently long to allow the algorithm to converge for either Batch or Stochastic Gradient Descent.

4. A larger learning rate, α, allows for quicker convergence (a larger "step" size in the direction of the lower gradient). However, a learning rate that is too large forces the update rate to follow the error, which can be very large, causing computation difficulties. Therefore, the learning rate must be a compromise between computation time vs computation capabilities. This will require special care as we increase k.

5. Note that, although the number of iterations it took for SGD to converge is larger than Batch GD (as seen in the plots), the calculations performed in each iteration are significantly lower. For Batch GD, the vector θ was computed using the error for the entire "batch" (i.e., all 70 samples) in each iteration, whereas in SGD, θ was computed using the error for each sample (i.e., a "batch" size of 1).

A technique commonly referred to as Mini-Batch Descent (MBD) provides a tradeoff between these two techniques. In MBD, the gradient descent is computed using a smaller subset of the entire batch (for example, 10 samples)

6. Figure 4 shows the hypothesis function for the three approaches (Normal, Stochastic, and Batch Gradient Descent) overlapped. As it can be seen, when run for sufficiently long iterations, SGD and BGD can approximate the Normal function very well.

7. For comparison, Figure 5 shows the plot for the hypothesis functions if the number of iterations is limited to 10,000, with $\alpha = 2e^{-5}$

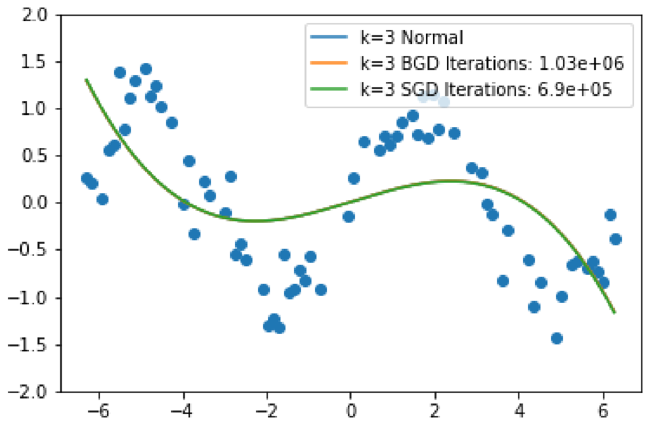

Figure 4: Normal, Stochastic, and Batch Gradient Descent with Polynomial Feature Mapping k = 3

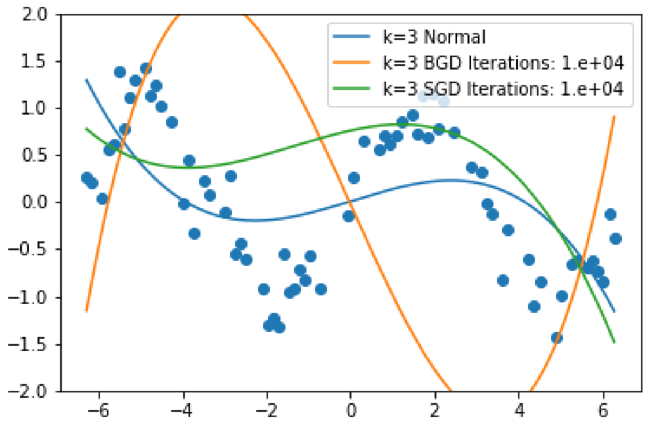

Figure 5: Normal, Stochastic, and Batch Gradient Descent with Polynomial Feature Mapping k = 3, at 10,000 iterations

1.4. Coding question: degree-k polynomial regression (using Normal, Batch GD and Stochastic GD)

Observations

1. Normal Equation (Figure 6): For models with $k = 5$ or $k = 10$, the Normal Equation captures the distribution of the dataset far better than $k = 3$, as hypothesized. However, at $k = 20$, the model "overfits" the dataset, i.e., it begins to model the noise within the dataset. This makes the model less "generalized" for use with validation, test, and operational datasets.

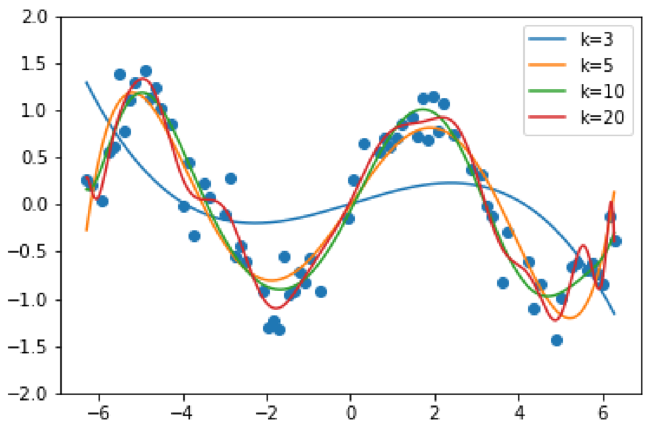

Figure 6: Normal Equation with Polynomial Feature Mapping k = [3, 5, 10, 20]

2. Gradient Descent (Figure 7) $k = 5$: Both Batch and Gradient descent models with $k = 5$ "fit" the dataset better, i.e., they capture the distribution of the dataset far better than $k = 3$, as hypothesized. This is likely because $k = 5$ allows the learning algorithm to map the dataset onto a feature-map which is more representative of the original distribution. It may be possible to achieve an even better fit with longer learning times (increasing iterations), higher feature mapping, k, and fine-tuning the learning rate, α.

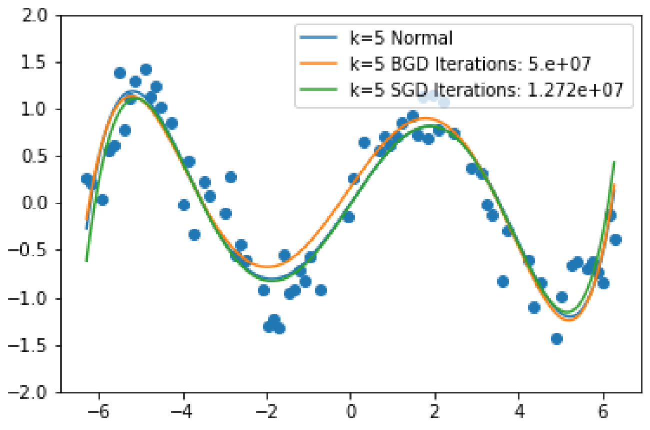

Figure 7: Normal, Stochastic, and Batch Gradient Descent with Polynomial Feature $k = 5$, $\alpha = 7e^{-7}$, and Convergence Threshold of $1e^{-9}$

3. As was observed in (1.3), as we in increase k, we must decrease the learning rate, α, in order to compute the loss $J(\theta)$ and the updated θ. This is because, as we increase the feature mapping k, the higher-power values grow exponentially, and the error values (hypothesis - target value) increase to very large numbers. Decreasing the learning rate ensures that we are *not* learning too much from these errors, and we do this by reducing our *step size* of the Gradient Descent. Additionally,

as we lower the learning rate, we must also lower the *Convergence Threshold*, which is the threshold at which the *change* in the value of θ is considered small enough for it to be considered *stabilized*.

However, a learning rate that is too low would result in insufficient learning at each timesetep, and would require a longer time to converge. Therefore, we must fine tune α to find the "ideal value". This can be avoided altogether by using the Normal Equation approximation.

4. Gradient Descent (Figure 8) $k = 10$: As we increase k, it becomes more and more challenging to fine tune the values of α and Convergence Threshold. Figure 8 shows the result of running the Batch and Stochastic GD algorithms for large iteration loops, with the Convergence Threshold set to $1e^{-9}$.

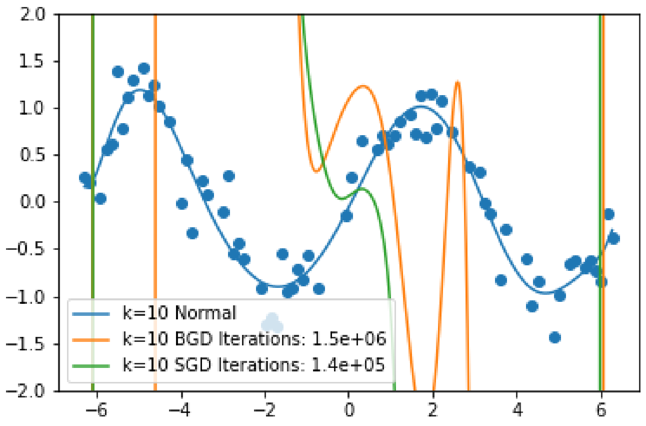

Figure 8: Normal, Stochastic, and Batch Gradient Descent with Polynomial Feature $k = 10$, $\alpha = 2e^{-15}$, and Convergence Threshold of $1e^{-9}$

Running the Gradient Descent algorithm for a longer time does not necessarily result in better convergence, as can be seen in Figure 9. In this figure, we ran the algorithm for *1 Billion* iterations (requiring 3 hours of compute time!), without achieving convergence, resulting in a poorly fitted model.

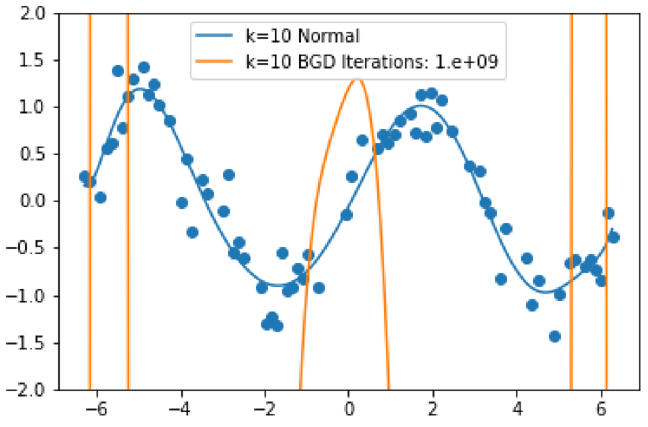

Figure 9: Normal, and Batch Gradient Descent with Polynomial Feature $k = 10$, $\alpha = 2e^{-15}$, and Convergence Threshold of $1e^{-15}$

5. Gradient Descent (Figure 10) $k = 20$: Similar to $k = 10$, as we increase k to $k = 20$, our GD models become harder to train. Figure 8 shows the result of running the Batch and Stochastic GD algorithms for large iteration loops.

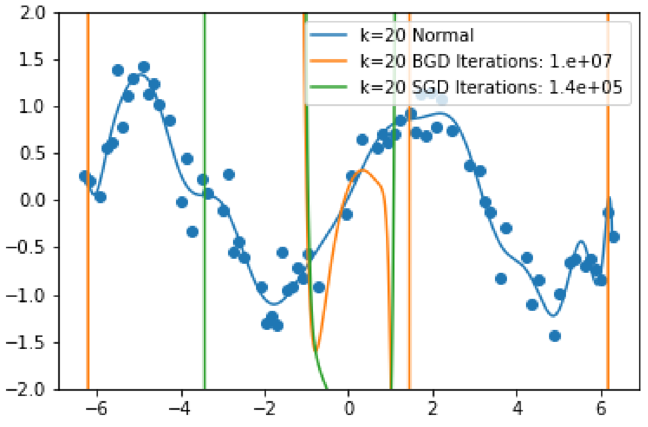

Figure 10: Normal, Stochastic, and Batch Gradient Descent with Polynomial Feature $k = 20$, $\alpha = 7e^{-35}$, and Convergence Threshold of $1e^{-30}$

Note, as mentioned previously, for $k = 20$, even the Normal Equation does not provide a good model. It "overfits" the data, i.e., tries to capture the noise in the dataset. This makes it less generalized for application on other datasets.

6. It can be seen that increasing the complexity of the feature mapping (for example, by using higher power mappings) may also increase the complexity of parameter tuning. Applying tech-

niques such as *Feature Scaling* and *Normalization* can help make the task of fine-tuning easier. However, based on experience in industry, for most practical applications, a simple estimation technique which is “good enough” is sufficient in most cases. Thus, in this case, using $k = 5$ would be sufficient for most practical applications, and at $k > 5$, we hit the “Point of Diminishing Return”

Note: Fortunately, having a closed form solution to the estimation problem, such as the Normal Equation for Linear Regression, eliminates the needs for complex fine tuning.

1.5. Coding question: other feature maps (using Normal, Batch GD and Stochastic GD)

Observations

1. Normal Equation: Figure 11 shows the results of fitting a sinusoudal function as part of the mapping feature, alongside polynomials, and using the Normal Equation. Since the original dataset was generated using a sine function, it can be seen that using a sine function for mapping fits the data very well, even for low values of k (i.e., $0 \leq k \leq 5$). In fact, it can be observed that the model fits the data very well for $k = 0$, i.e., mapping only on the sinusoidal function $sin(x)$.

Furthermore, it can be seen that for large values of k (i.e., $k \geq 10$), the model “overfits” the data, i.e., starts to represent the noise within the dataset.

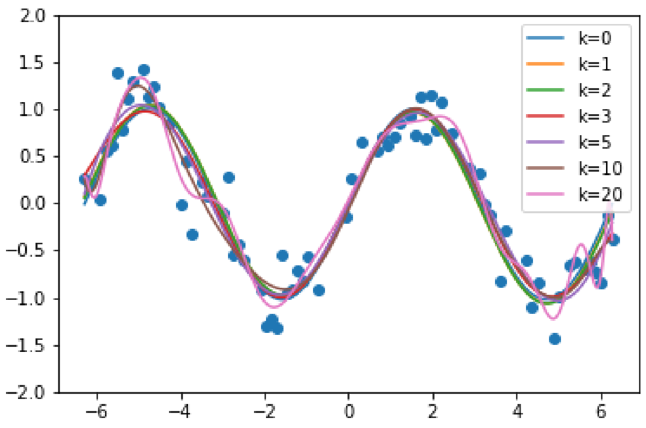

Figure 11: Normal Equation with Feature Mapping to $sin(x)$ and Polynomials $k = [0, 1, 2, 3, 5, 10, 20]$

2. Batch Gradient Descent: Figure 12 shows the results of fitting a sinusoudal function as part of the mapping feature, alongside polynomials, and using Batch Gradient Descent. Similar to the Normal Equation, the models fit the data very well, even for low values of k (i.e., $0 \leq k \leq 5$). In fact, it can be observed that the model fits the data very well for $k = 0$, i.e., mapping only on the sinusoidal function $sin(x)$.

Similar to the Normal Equation, it can be seen that for large values of k (i.e., $k \geq 10$), the model “overfits” the data, i.e., starts to represent the noise within the dataset.

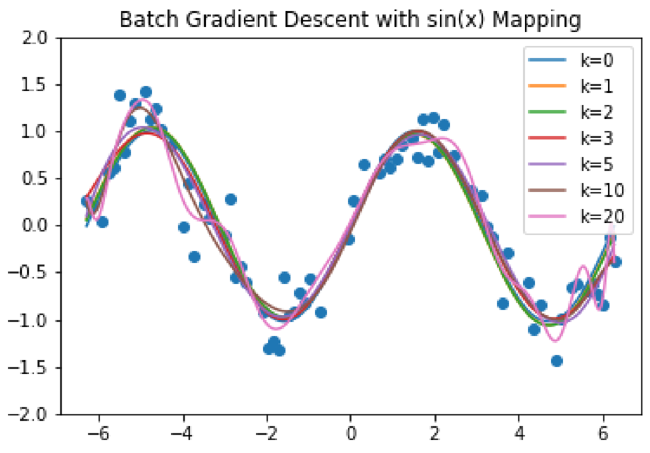

Figure 12: Batch Gradient Descent with Feature Mapping to $sin(x)$ and Polynomials $k = [0, 1, 2, 3, 5, 10, 20]$

3. Stochastic Gradient Descent: Figure 13 shows the results of fitting a sinusoudal function as part of the mapping feature, alongside polynomials, and using Stochastic Gradient Descent.

Although the model does not fit the dataset as well as Batch Gradient Descent for $k \geq 5$ (due to less time spent parameter tuning), it still fits the data far better than without sinusoudal mapping for $0 \leq k \leq 3$. In fact, it can be observed that the model fits the data very well for $k = 0$, i.e., mapping only on the sinusoidal function $sin(x)$.

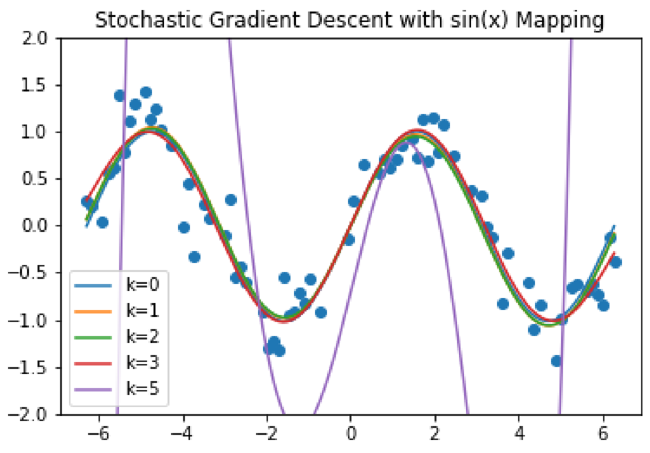

Figure 13: Stochastic Gradient Descent with Feature Mapping to $sin(x)$ and Polynomials $k = [0, 1, 2, 3, 5]$

1.6. Overfitting with expressive models and small data

Observations

1. Normal Equation: Figure 14 shows the results of fitting a Normal Equation on the Small dataset. It can be seen that the number of samples are insufficient for proper model training

- For values of $k \leq 2$, the model does not sufficiently learn, resulting in an "underfitted" model. Such a model is one where not enough learning has taken place, and thus does would not provide reasonable accuracy on real dataset predictions.
- For values of $k \geq 5$, the model passes through each individual point, resulting in an "overfitted" model. This model is not generalized, and would not provide reasonable accuracy on a real dataset predictions.
- For values of $k = 3$, the model represents the original (test) dataset reasonably well. However, this model would still not be sufficient for reasonably accurate predictions on a real dataset because there are not enough data samples.

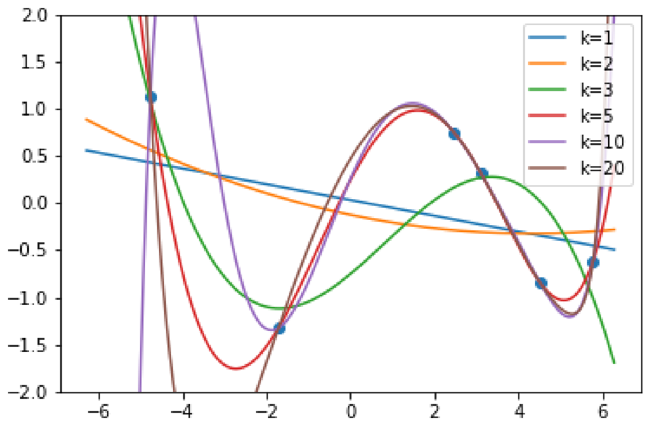

Figure 14: Normal Equation on Small Dataset with Feature Mapping to Polynomials $k = [1, 2, 3, 5, 10, 20]$

2. Batch Gradient Descent: Figure 15 shows the results of fitting a Batch Gradient Descent model on the Small dataset. The result is very similar to using the Normal Equation.

- For values of $k \leq 2$, the model does not sufficiently learn, resulting in an "underfitted" model.
- For values of $k \geq 5$, the model seems to be "overfitted", modeling the noise in the data.
- For values of $k = 3$, the model represents the original dataset reasonably well. However, this model is still not sufficient for reasonably accurate predictions on a real dataset.

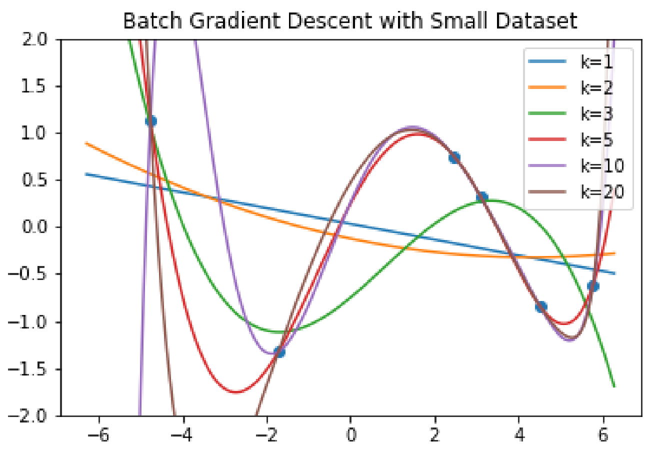

Figure 15: Batch Gradient Descent on Small Dataset with Feature Mapping to Polynomials $k = [1, 2, 3, 5, 10, 20]$

2. Incomplete, Positive-Only Labels

In Q2, we are dealing with the problem of classifying partially visible positive-only labels, i.e., determining the category in which a sample belongs when we only have partial-visibility into the output, and only when the target is positive.

We will be using Logistic Regression to solve the classification problem and create a supervised-learning model that is able to learn from training data. The method of implementation selected for this section is Newton's Method. At the end of this report, we re-visit some of these questions using the Gradient Descent implementation, and compare the findings.

In subsequent sections, we will use the following terms:

- t-label: True, fully visible target variables
- y-label: Partially visible target variables
- Training Dataset: The dataset containing the training data, including t-labels, y-labels, and features. The model is trained using this dataset.
- Test Dataset: The dataset containing the test data, including t-labels, y-labels, and features. The model is evaluated using this dataset.
- Validation Dataset: The dataset containing the validation data, including t-labels, y-labels, and features. In section 2.6, the model is "adjusted" using this dataset
- Decision Boundary: The line at which the model separates the positives from the negatives. The line, shown in red on the plots below, represents the points where the predicted probability is 0.5.

- Accuracy: A measure of the total accurate predictions as a ratio of total test samples, i.e.:

$$Accuracy = \frac{TotalCorrectPredictions}{TotalSamplesInTestSet}$$

  It should be noted that Accuracy is not the only relevant metric of evaluating model performance in the real world, but it is good enough for this problem.

- Confusion Matrix: A table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It visually depicts the comparison between the true and predicted categories. Figure 16 shows what a Confusion Matrix looks like for 2-class classification.

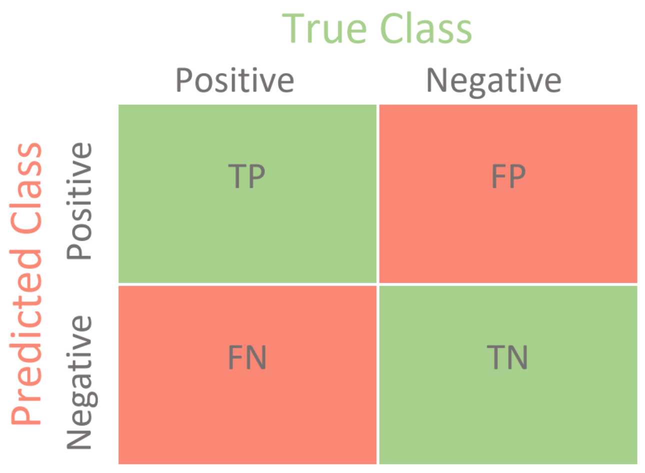

Figure 16: Confusion Matrix Template

2.1. Coding problem: ideal (fully observed) case

First, we **train the model on the t-labels, i.e., fully visible target variables, of the Training dataset**. This should give us the *best* possible model, since it is trained on the complete ground-truth data.

Figure 17 shows how the model fits the t-labels Training Dataset. This is the data the model is trained on. As expected, it disects the training dataset quite well, as shown by the decision boundary.

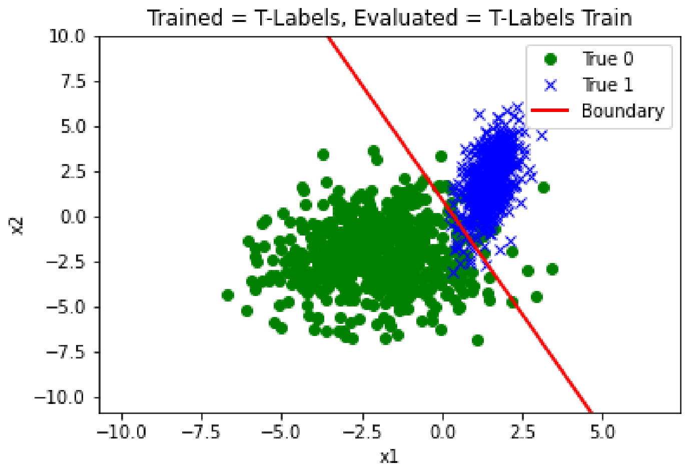

Figure 17: Evaluating Model on t-labels of Training Dataset

Figure 18 shows the results of evaluating the model against t-labels of the Test Dataset. Since the data was trained on the "best case scenario" dataset (i.e., t-labels of the training dataset), it can be seen that the model classifies the test data very well, as shown by the decision boundary in red.

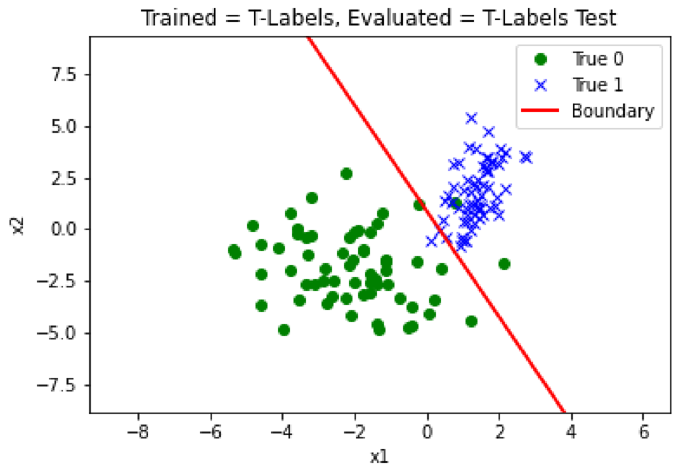

Figure 18: Evaluating Model on t-labels of Test Dataset

Figure 19 shows the Confusion Matrix, along with the accuracym for Figure 18. This is the best possible accuracy that can be achieved using this classification technique on this dataset. In subsequent sections, we will try to build a Logistic Regression classification model which matches this accuracy, but using *only* the y-labels (partially hidden labels) of the Training Dataset.

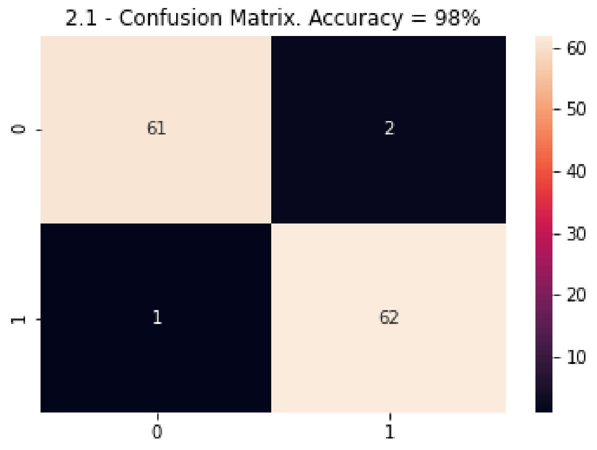

Figure 19: Confusion Matrix and Accuracy - "Best Case Scenario"

Finally, Figure 20 plots the comparison between the real t-labels of the Test Dataset, and the predictions of our model. As can be seen by a visual inspection, there is no clear-cut classification boundary between the 0 and 1 labels amongst the t-labels, and there are several labels that overlap each other. This tells us that it may be impossible to classify these points with 100% accuracy in 2-Dimensions.

There are classification techniques, such as ***Support Vector Machines*** that can be used to map the points into higher dimensions, which may be used in order to achieve higher accuracy.

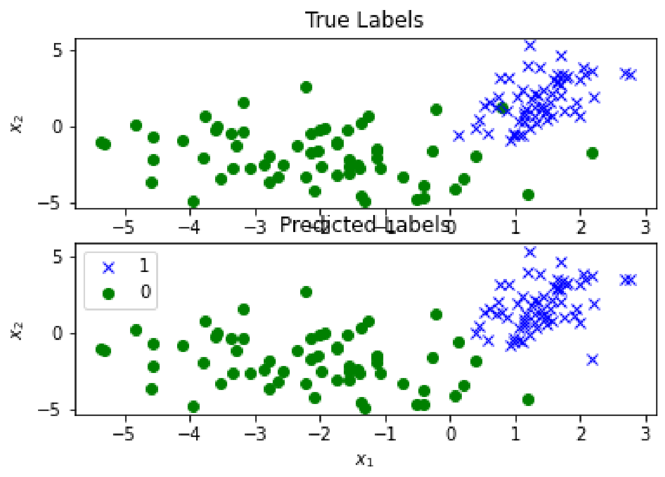

Figure 20: True vs Model Predicted t-labels

2.2. Coding problem: The naive method on partial labels

In this section, we **train the model on the y-labels of the Training dataset**. This will emulate the realistic situation, where we are trying to predict the class based only on partially-visible data.

Figure 21 shows the results of evaluating the model against t-labels of the Test Dataset. Since the data was trained on the partially visible dataset, it can be seen that the model does not fit the data very well, and is highly biased towards classifying samples into the category 0. This is further exemplified in Figure 22.

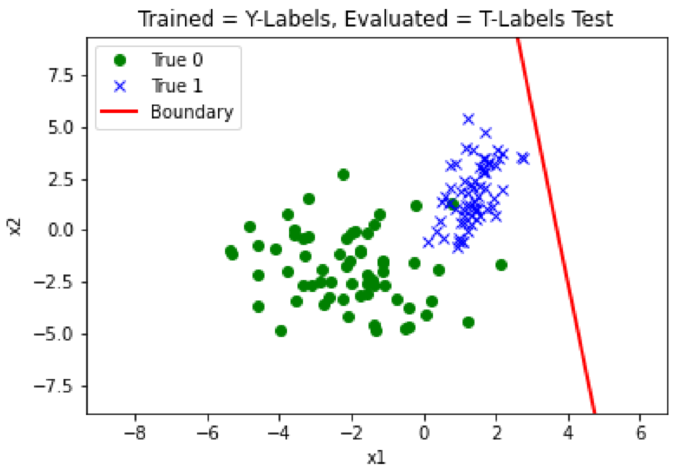

Figure 21: Model Evaluation against t-labels of the Test Dataset

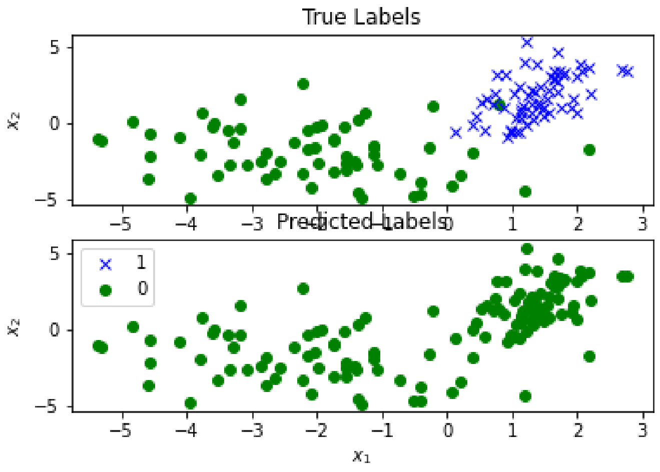

Figure 22: True vs Model Predicted t-labels

Figure 23 shows the Confusion Matrix, along with the accuracy, for this model. It is obvious that this classification-model is not a good classifier when trained purely on the partially-visible target variables, and classifies all samples as 0.

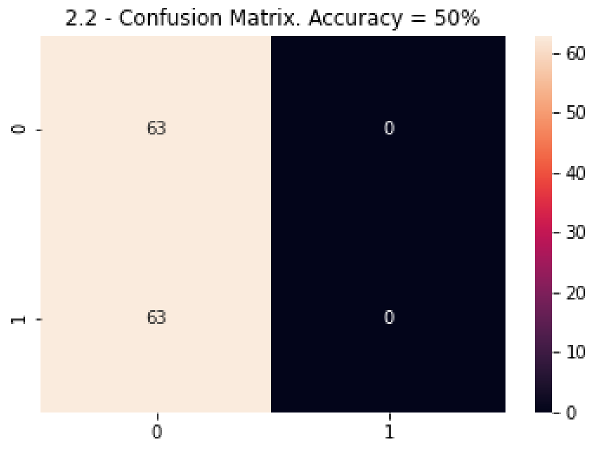

Figure 23: Confusion Matrix and Accuracy - "Not Very Good"

2.3. Warm-up with Bayes Rule

We start by using Bayes' Rule to write $p(t^{(i)} = 1|y^{(i)} = 1, x^{(i)})$ as a fraction.

$$p(t^{(i)} = 1|y^{(i)} = 1, x^{(i)}) = \frac{p(y^{(i)} = 1|t^{(i)} = 1, x^{(i)})p(t^{(i)} = 1|x^{(i)})}{p(y^{(i)} = 1|x^{(i)})} \tag{0.3}$$

We can rewrite the denominator using the Total Prbability Rule as:

$$p(y^{(i)} = 1|x^{(i)}) = p(y^{(i)} = 1, t^{(i)} = 1|x^{(i)}) + p(y^{(i)} = 1, t^{(i)} = 0|x^{(i)})$$

We then use the Probability Chain Rule (general product rule) to expand the denominator.

$$= \frac{p(y^{(i)} = 1|t^{(i)} = 1, x^{(i)})p(t^{(i)} = 1|x^{(i)})}{p(y^{(i)} = 1, t^{(i)} = 1|x^{(i)}) + p(y^{(i)} = 1, t^{(i)} = 0|x^{(i)})} \tag{0.4}$$

$$= \frac{p(y^{(i)} = 1|t^{(i)} = 1, x^{(i)})p(t^{(i)} = 1|x^{(i)})}{p(y^{(i)} = 1|t^{(i)} = 1, x^{(i)})p(t^{(i)} = 1|x^{(i)}) + p(y^{(i)} = 1|t^{(i)} = 0, x^{(i)})p(t^{(i)} = 0|x^{(i)})} \tag{0.5}$$

We then substititute the observation rules provided in the question, namely:

$$\forall x \begin{bmatrix} p(y^{(i)} = 1|t^{(i)} = 1; x^{(i)} = x) = \alpha \\ p(y^{(i)} = 1|t^{(i)} = 0; x^{(i)} = x) = 0 \end{bmatrix}$$

$$= \frac{\alpha \cdot p(t^{(i)} = 1|x^{(i)})}{\alpha \cdot p(t^{(i)} = 1|x^{(i)}) + 0 \cdot p(t^{(i)} = 0|x^{(i)})} \tag{0.6}$$

$$= \frac{\alpha \cdot p(t^{(i)} = 1|x^{(i)})}{\alpha \cdot p(t^{(i)} = 1|x^{(i)})} \tag{0.7}$$

Hence:

$$\boxed{\boldsymbol{p(t^{(i)} = 1|y^{(i)} = 1, x^{(i)}) = 1}} \tag{0.8}$$

2.4. CP8318 Only Question

We first expand $p(t^{(i)}=1|x^{(i)})$ using the Total Probability Rule:

$$p(t^{(i)}=1|x^{(i)}) = p(t^{(i)}=1, y^{(i)}=1|x^{(i)}) + p(t^{(i)}=1, y^{(i)}=0|x^{(i)}) \tag{0.9}$$

Re-arrange the equation using $p(A \cap B) = p(A, B) = p(B, A)$

$$= p(y^{(i)}=1, t^{(i)}=1|x^{(i)}) + p(t^{(i)}=1, y^{(i)}=0|x^{(i)}) \tag{0.10}$$

We then use Bayes' Rule to expand the terms:

$$= p(t^{(i)}=1|y^{(i)}=1, x^{(i)})p(y^{(i)}=1|x^{(i)}) + p(y^{(i)}=0|t^{(i)}=1, x^{(i)})p(t^{(i)}=1|x^{(i)}) \tag{0.11}$$

Then, we substitute in the observation rules provided in the question, namely:

$$\forall x \begin{bmatrix} p(y^{(i)}=0|t^{(i)}=1; x^{(i)}=x) = 1-\alpha \\ p(y^{(i)}=0|t^{(i)}=0; x^{(i)}=x) = 1 \end{bmatrix}$$

$$p(t^{(i)}=1|x^{(i)}) = 1 \cdot p(y^{(i)}=1|x^{(i)}) + (1-\alpha) \cdot p(t^{(i)}=1|x^{(i)}) \tag{0.12}$$

Re-arranging the right side of the equality gives us the equation:

$$\boxed{\boldsymbol{p(t^{(i)}=1|x^{(i)}) = \frac{1}{\alpha} p(y^{(i)}=1|x^{(i)})}} \tag{0.13}$$

2.5. CP8318 Only Question: Estimating α

By definition, our hypothesis function is:

$$h(x^{(i)}) = p(y^{(i)}=1|x^{(i)}) \tag{0.14}$$

Writing out the expected value of $h(x^{(i)})$:

$$\mathbb{E}[h(x^{(i)})|y^{(i)}=1] = \mathbb{E}[p(y^{(i)}=1|x^{(i)})|y^{(i)}=1] \tag{0.15}$$

Expanding the right side of the equality using Bayes' Rule:

$$= \mathbb{E}[p(y^{(i)}=1|t^{(i)}=1, x^{(i)})p(t^{(i)}=1|x^{(i)}) \tag{0.16}$$
$$+ p(y^{(i)}=1|t^{(i)}=0, x^{(i)})p(t^{(i)}=0|x^{(i)})|y^{(i)}=1] \tag{0.17}$$

Substituting in the observation rules provided in the question:

$$= \mathbb{E}[\alpha \cdot p(t^{(i)}=1|x^{(i)}) + 0|y^{(i)}=1] \tag{0.18}$$
$$= \alpha \cdot \mathbb{E}[p(t^{(i)}=1|x^{(i)})|y^{(i)}=1] \tag{0.19}$$

In section (2.3), we proved that:

$$\begin{aligned}\mathbb{E}[p(t^{(i)}=1|x^{(i)})|y^{(i)}=1] &= p(t^{(i)}=1|y^{(i)}=1,x^{(i)}) && (0.20)\\ &= 1 && (0.21)\end{aligned}$$

Thus:

$$\boxed{\mathbb{E}[h(x^{(i)})|y^{(i)}=1]=\alpha} \quad (0.22)$$

2.6. Coding Problem: Applying Adjustment Factor α, Using Newton's Method for Minimizing $J(\theta)$

In this section, we **train the model on the y-labels of the Training dataset, and apply a Scaling Factor to the model's predictions**. This will emulate the realistic situation where we are trying to predict the class based only on partially-visible data. We will leverage the proof presented in section (2.4):

$$p(t^{(i)}=1|x^{(i)})=\frac{1}{\alpha}p(y^{(i)}=1|x^{(i)}) \quad (0.23)$$

From the above equation, we see that we can transform y-label predictions into t-label predictions by simply dividing the output probability by a factor, α.

We calculate α using the approximation presented in the Assignment:

$$\alpha \approx \frac{1}{|V_+|}\sum_{x^{(i)}\in V_+} h(x^{(i)}) \quad (0.24)$$

where:
V_+: Is the vector containing only the positive t-labels of the Validation Dataset.
$h(x^{(i)})$: Is the hypothesis vector containing the predicted probabilities for the target variable, y-label.

We will employ the following steps to achieve a better performing classification model:

1. Train the model on y-labels of the Training Dataset
2. Evaluate the model on the Validation and Test Datasets
3. Calculate the scaling factor, α, using the Validation Dataset
4. Make predictions for y-labels using the Test Dataset, and apply the scaling factor to obtain t-labels, i.e., ground-truth predictions.
5. Evaluate the final predictions against the "best case scenario" classifier described in section (2.1)

Prior to the Application Adjustment Factor α

Figure 24 and 25 show the results of evaluating the model against y-labels of the Validation Dataset and t-labels of the Test Dataset, respectively, *before* any correction factor is applied. It can be seen that the decision boundary is not where it should be to classify the points correctly.

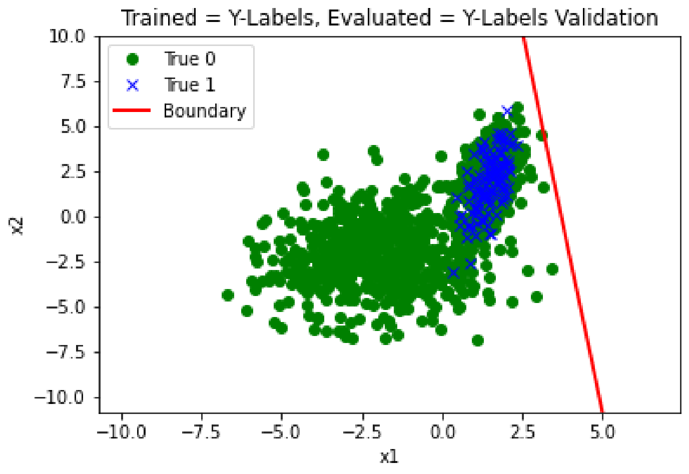

Figure 24: Model Evaluation Against y-labels of the Validation Dataset, before Adjustment

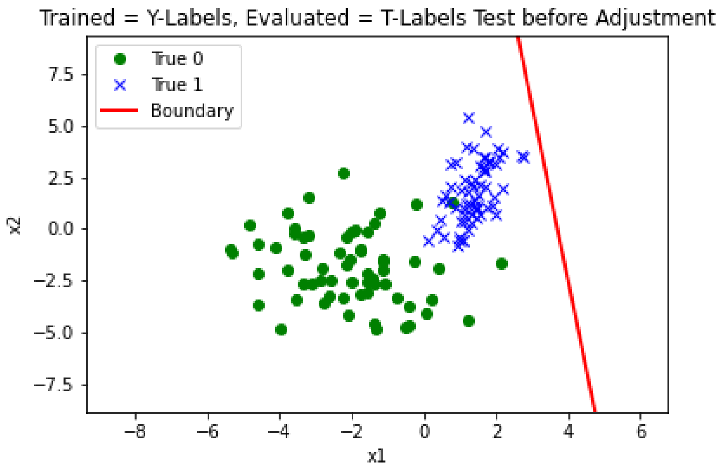

Figure 25: Model Evaluation Against t-labels of the Test Dataset, before Adjustment

On close inspection, it is can be seen that the decision boundary has a slope somewhat close to the "best case scenario" presented in section (2.1). In fact, it is apparent that if the decision boundary is simply translated to the "left" the "correct amount", it would do a much better job of classification (depicted in Figure 26).

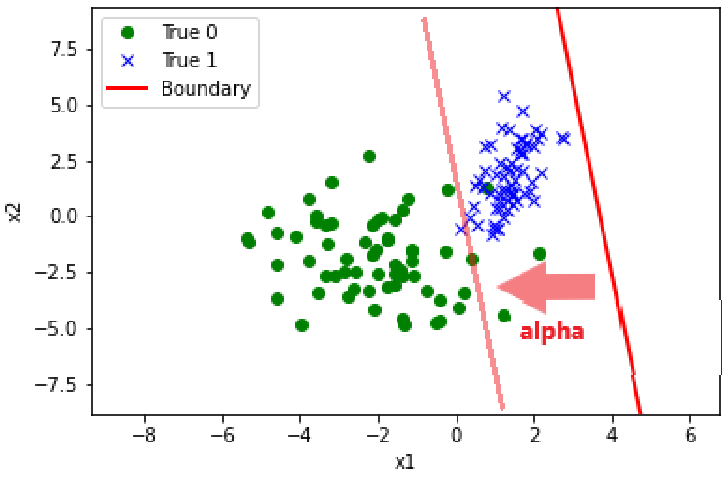

Figure 26: The Model may be improved by translating the Decision Boundary

This "correct amount" of translation is determined by α. We use the known t-labels and y-labels in the Validation Dataset to calculate the scaling factor α, which we then apply to the predictions $h(x^{(i)})$ for the Test Dataset, in order predict the ground-truth target, t-label.

After the application of the Adjustment Factor α

Figure 27 shows the result of evaluating the model against t-labels of the Test Dataset, *after* the correction factor α is applied. It can be seen that the decision boundary now does a much better job of classifying the points.

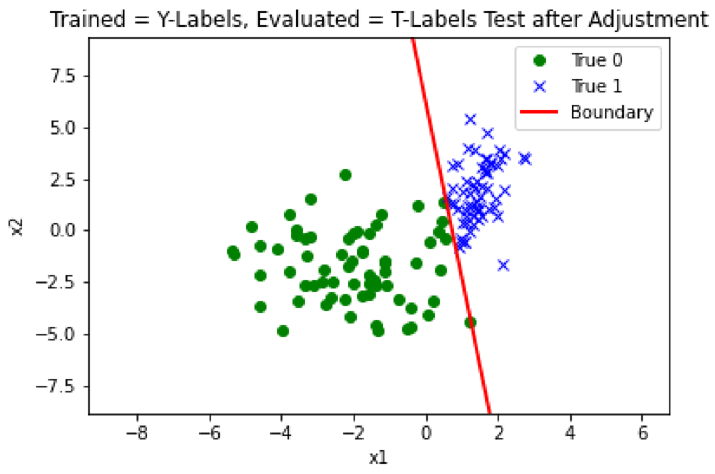

Figure 27: Model Evaluation Against t-labels of the Test Dataset, after Adjustment

Figure 28 compares the predicted t-labels with the true t-labels of the Test Dataset. It can be visually confirmed that the model performs far better after scaling is applied. This is further confirmed by the Confusion Matrix and Accuracy measurements shown in 29.

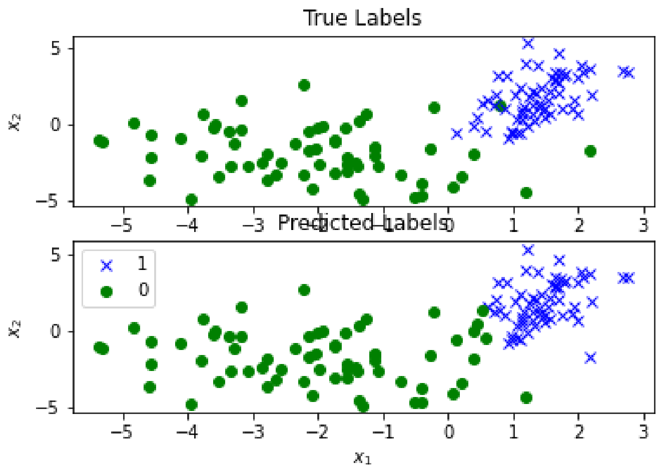

Figure 28: True vs Model Predicted t-labels, after Adjustment

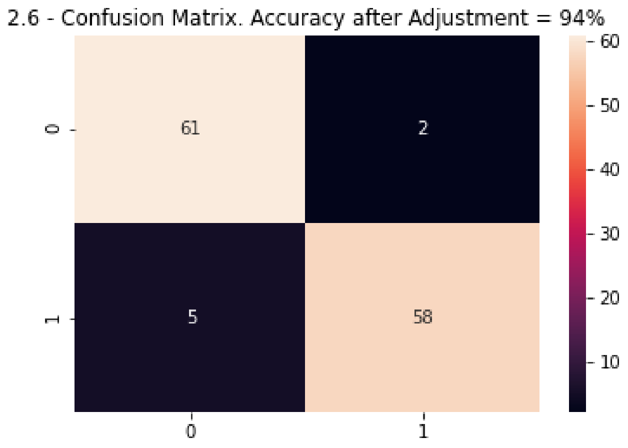

Figure 29: Confusion Matrix and Accuracy - after Adjustment

Thus, by applying the correction factor calculated using the Validation Dataset, we have achieved an accuracy very close to the "best case scenario" presented in section (2.1). This is a profound result, as it means that we can classify only partially-visible samples quite well, if we have access to a representative dataset that provides a mapping from partial to fully visible target variables.

Revisiting 2.1 to 2.6. Coding Problem: Using Gradient Descent for Minimizing $J(\theta)$

In the previous sections, we used Newton's Method to determine the final value of θ, determined by minimizing the loss function $J(\theta)$. In this section, we will perform the same minimization using Gradient Descent, and compare the results.

Figures 30 and 31 shows the result of re-performing the tasks in section (2.6) using Gradient Descent. As expected, the classification accuracy result is exactly the same.

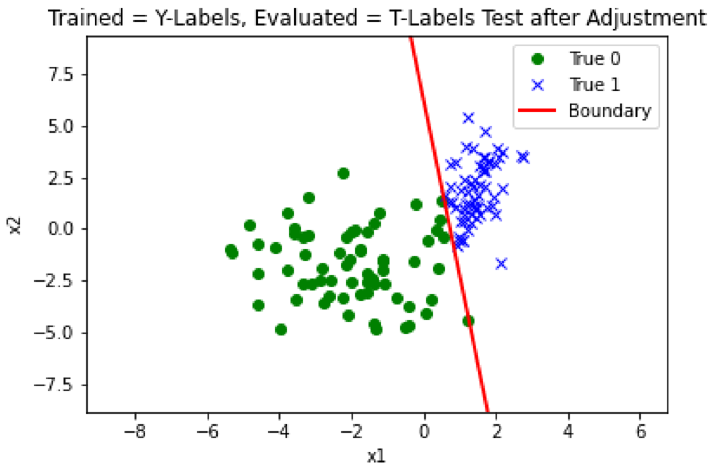

Figure 30: Model Evaluation Against t-labels of the Test Dataset, after Adjustment - Gradient Descent

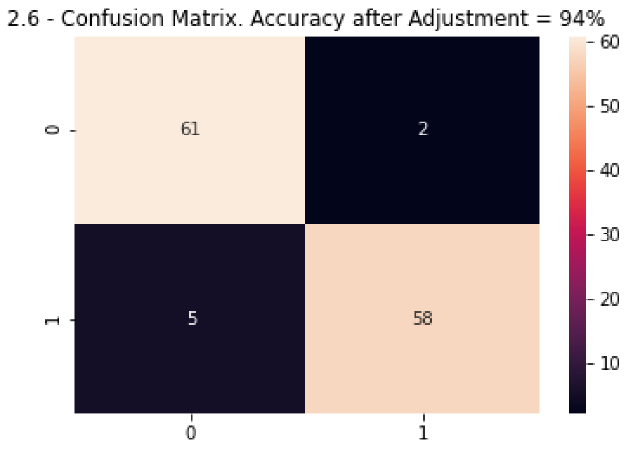

Figure 31: Confusion Matrix and Accuracy - after Adjustment - Gradient Descent

The key difference in the two approaches, however, is the time of execution required for model fitting.

- The execution time until convergence for Newton's Method, averaged over 10 runs, was 1.3s and 420 iterations.
- The execution time until convergence for the Gradient Descent Algorithm, averaged over 10 runs (using a Learning Rate of 0.3), was 0.4s and 1150 iterations.

Thus, on a generic commercial laptop, for the given dataset, with θ initialized as a vector of zeros, the Gradient Descent Method requires *3 times less time* to compute despite requiring *3 times* *more iterations* that Newton's Method.

This difference is because of the fact that the Gradient Descent method requires only the Gradient to be calculated at each iteration, whereas Newton's Method requires us to calculate *both* the Gradient and the Hessian to compute the θ *update rule.* Thus, the Gradient Descent algorithm requires less computation at each step, and is typically superior to Newton's Method in terms of computation time *for most sufficiently large datasets.*

Alternatively, because Newton's Method bases the θ update on both the Gradient and the Hessian, it provides, on average, a more accurate update rule in the direction of the minima at each timestep. In other terms, in addition to looking at the locally optimal direction (Gradient), it also looks at information about the curvature of the function (Hessian) to determine the next step. Thus, Newton's Method is typically able to converge in *fewer iterations* than Gradient Descent. This is especially useful for smaller datasets, when there is not enough information to learn using Gradient Descent, or on datasets with many features.