CIS 419/519: Applied Machine Learning — Fall 2024

Homework 2

Handed Out: October 2 — *Due: 7:59 pm October 22*

Name: Alan Wu

PennKey: alanlwu

PennID: 41855518

1 Declaration

- **Person(s) discussed with:** *Your answer*
- **Affiliation to the course: student, TA, prof etc.** *Your answer*
- **Which question(s) in coding / written HW did you discuss?** ***Your answer***
- **Briefly explain what was discussed.** *Your answer*

2 Multiple Choice & Written Questions

1. (a) Our unregularized logistic regression decision boundary will probably be a line with some intercept (positive or negative) on the x1 or x2 axis. There will be 0-1 misclassified points. In our example, we have 2 misclassified points. Here is a simple example:

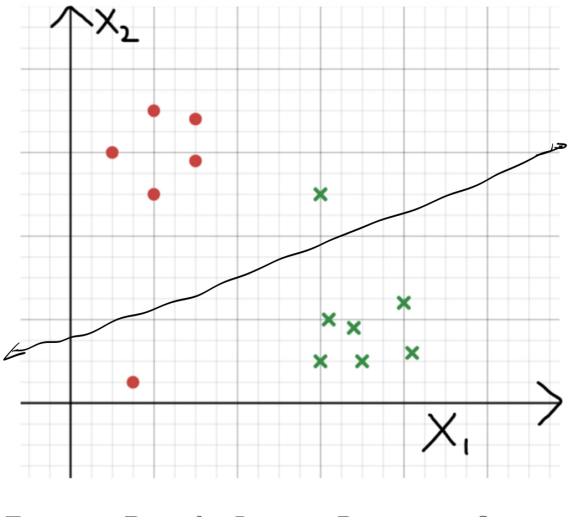

Figure 1: Unregularized Logistic Regression Decision Boundary

(b) When we apply heavy regularization to the θ_0 parameter, we can see that it will be pushed to 0. This means that the decision boundary will go through the origin, as when $x_1 = x_2$, the decision boundary will be 0 and the predicted probability of either class is 0.5. There should be 1-2 misclassified points. In our example, we have 1 misclassified point. And thus, our figure looks like this:

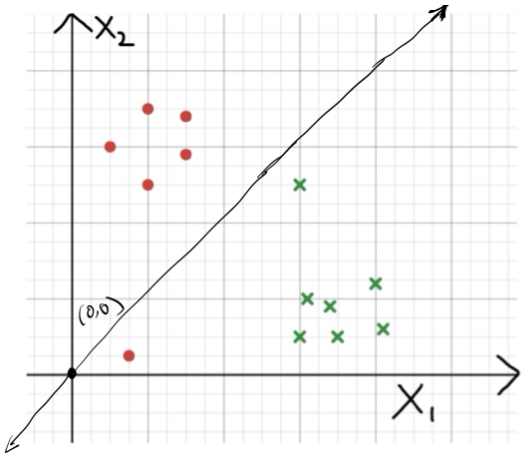

Figure 2: Heavy Regularized Logistic Regression on θ_0 Decision Boundary

(c) When we apply heavy regularization to the θ_1 parameter, then that means that the θ_1 parameter will be pushed to 0. Since we have the line as $\theta_0 + \theta_2 x_2$, then we know that we have the decision boundary $x_2 = -\theta_0/\theta_2$. Our decision boundary will be a horizontal line that splits the clusters into two on the x_2 axis. There should be 1-2 misclassifications. In our figure we have 2 misclassified points. Here is the figure:

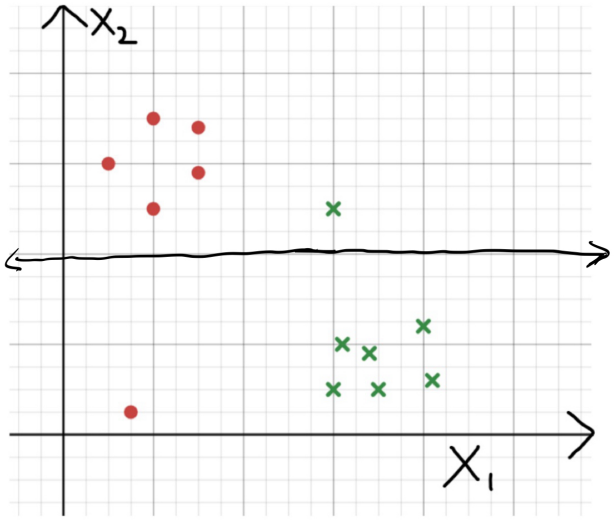

Figure 3: Heavy Regularized Logistic Regression on θ_1 Decision Boundary

(d) When we apply heavy regularization to the θ_2 parameter, the θ_2 parameter will be pushed to 0. In this case, our decision boundary will follow the line $\theta_0 + \theta_1 x_1$. Thus, the resulting decision boundary will be $x_1 = \theta_0/\theta_1$, which will be a vertical line that runs through the x_1 axis. Our sample boundary has 0 misclassified points. Here is the figure:

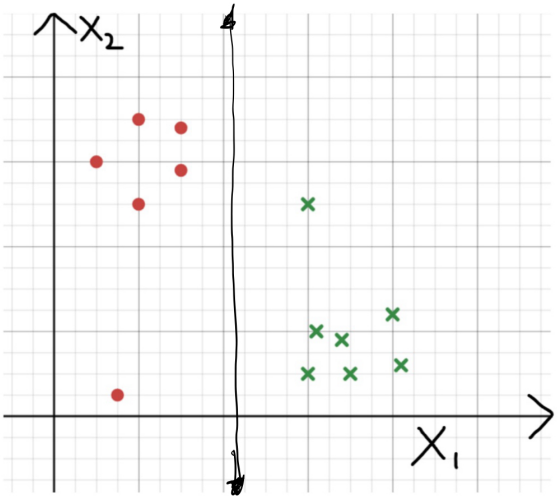

Figure 4: Heavy Regularized Logistic Regression on θ_2 Decision Boundary

2. (a) If we have two training points and we are using a k = 1 neighbors algorithm, then the decision boundary will be the line that is the perpendicular bisector of the midpoint of the two training points. We already know that k nearest neighbors algorithm will have decision boundary halfway through the two points because that point is equally distant from both points. Below details an example of this:

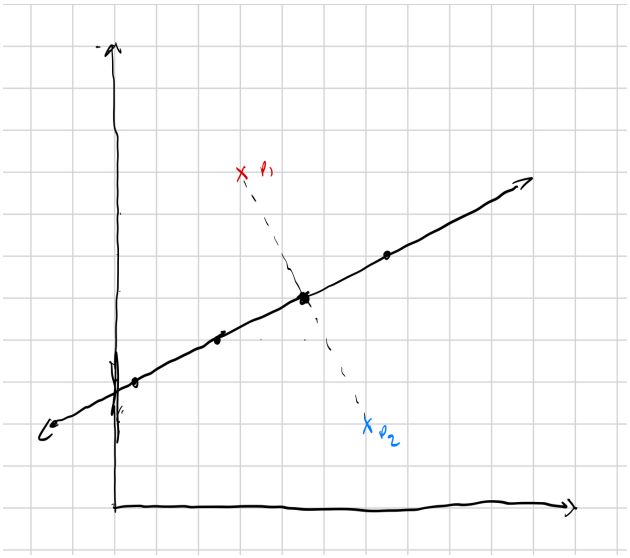

Figure 5: Decision Boundary K = 1

(b) Yes, it can explain any dataset. It will be completely determined by the training data.

(c) As k approaches ∞, the decision boundary will become smoother. The resulting model function family will be the majority class of the training set, as when we use infinite point, we are essentially using the entire training set.

(d) Increasing the k will increase the bias and lower the variance. As we have more points to reference in our regression or classification, we will have a smoother decision boundary. Since there are a greater amount of points, the average of these points will be more representative of the true value of any given point. The bias increases because as we have more points, the average prediction will be closer to the true value.

(e) In KNN classification, we generally take the majority label and call it our prediction. We can adjust this for a given accuracy/recall/precision metric by simply changing how many values out of k we commit to a positive or negative class. Another approach we could use is to measure the probability of a given class by counting the number of positive and negative outcomes, and then determining the class prediction based on a threshold that we tune.

3. (a) We may expect that when we use post-pruning, we may not need early stopping. To clarify that this is indeed not the case, we can first consider what each of the concepts means and then see how they are related. Post pruning occurs after we've finished building out the entire tree and then we begin to take out subsets of the tree and replace them as leaves if they do not improve the performance of the tree. We will keep pruning until taking away more tree subsets yields us negative validation set gains. On the other hand, early stopping will stop us from building out the entire tree. This means that perhaps some features will not be split on or we will not have the full tree. In this way, it becomes clear why we

could possibly need both. When we strictly use post-pruning, we are going to be post-pruning from a tree built out with all of its features. Given this fact, if some of the features are significantly not important, then we may still include them in the tree. We will prune the tree to a certain extent, but these features may already be overfitted to the noise in the training data. However, if we combine the early stopping and the post-pruning, we will start pruning with a tree that is well-fitted based on only the most significant features (as determined by ourselves) and then prune further to prevent our decision tree from overfitting.

(b) We are given the following conditions for when increasing the minimum samples needed to split will increase bias/variance:

i. increase variance
ii. increase bias
iii. increase variance
iv. increase variance

4. (a) We know the computation of the entropy of a given dataset is going to be $H(z) = -\sum_y P(Y = y) \log_2 P(Y = y)$
Therefore, we can compute the entropy of the dataset as follows:

There are 3 positive samples and 1 negative sample thus

$$\begin{aligned} H(z) &= -\sum_y P(Y = y) \log_2 P(y = y) \\ &= -[P(Y = 1) \log_2 P(Y = 1) + P(Y = 0) \log_2 P(Y = 0)] \\ &= -[0.75 \log_2(0.75) + 0.25 \log_2(0.25)] \\ &= 0.811 \end{aligned}$$

Therefore, the entropy of the current training samples is 0.811.

(b) The information gain is going to be the entropy of the parent node minus the weighted sum of the entropy of the child nodes. We will have to first define the computation for each of the features a1 and a2:

$IG(Z, a1) = H(Z) - H(Z[a1 = 1])P(a1 = 1) - H(Z[a1 = 0])P(a1 = 0)$ and

$IG(Z, a2) = H(Z) - H(Z[a2 = 1])P(a2 = 1) - H(Z[a2 = 0])P(a2 = 0)$

We will compute the individual information gain for each of the features, first for a1. We will use that there are 2 samples where feature a1 is positive and 2 where a1 is negative. Of all the samples for a1, 1 of them is negative and 3 samples are positive.

$$
\begin{aligned}
&IG(Z, a1) = H(Z) - H(Z[a1 = 1])P(a1 = 1) - H(Z[a1 = 0])P(a1 = 0) \\
&= 0.811 + [P(Y = 1|a1 = 1) \log_2 P(Y = 1|a1 = 1) \\
&+ P(Y = 0|a1 = 1) \log_2 P(Y = 0|a1 = 1)]P(a1 = 1)+ \\
&[P(Y = 1|a1 = 0) \log_2 P(Y = 1|a1 = 0) + P(Y = 0|a1 = 0) \log_2 P(Y = 0|a1 = 0)]P(a1 = 0) \\
&= 0.811 + [1 \log_2 1 + 0 \log_2 0]0.5 + [0.5 \log_2 0.5 + 0.5 \log_2 0.5]0.5 \\
&= 0.811 - log_2 0.5 \\
&= 0.311
\end{aligned}
$$

The information gain of the dataset with respect to feature a1 is 0.311.

Let's compute the individual information gain for a2: Note that for a2, there are 3 samples where feature a2 is positive and 1 sample where a2 is negative (with respect to itself). Of all the samples for a2, 3 of them are classified as positive and 1 of them is classified as negative.

$$
\begin{aligned}
&IG(Z, a2) = H(Z) - H(Z[a2 = 1])P(a2 = 1) - H(Z[a2 = 0])P(a2 = 0) \\
&= 0.811 + [P(Y = 1|a2 = 1) \log_2 P(Y = 1|a2 = 1) \\
&+ P(Y = 0|a2 = 1) \log_2 P(Y = 0|a2 = 1)]P(a2 = 1)+ \\
&= 0.811 + [0.667 \log_2 0.667 + 0.333 \log_2 0.333]0.75 + [1 \log_2 1 + 0]0.25 \\
&= 0.811 - (0.918) * 0.75 \\
&= 0.811 - 0.689 \\
&= 0.122
\end{aligned}
$$

The information gain of the dataset with respect to feature a2 is 0.122.

(c) The figure:

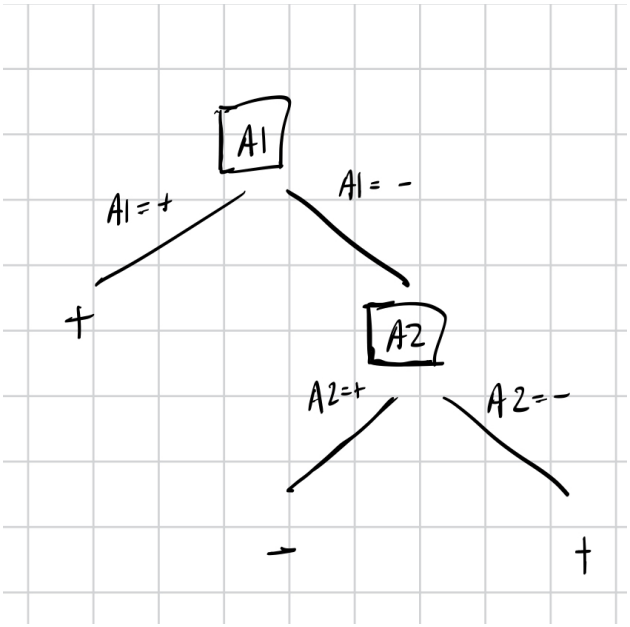

Figure 6: Decision Tree

5. (a) K Nearest Neighbors algorithm assumes that the predictions will be close in some arbitrary distance function. In this way, for our problem, since we are trying to predict the binary outcome of a house price, then we are assuming that houses that have similar prices are similar in square footage and also the number of bedrooms.

   (b) Euclidean distance will be a bad metric given our two features: square footage and the number of bedrooms because the two features are not on the same scale. The number of bedrooms is a discrete variable that takes on low values of 1, 2, 3, etc. while the square footage of the house is a continuous variable that can take on any value, usually in the thousands. If we use the variables as they are, the square footage will dominate the distance computation. What we can do to solve the issue is normalize both variables on the same scale (say on the z-score scale or limit it to 0 to 1).

   (c) The type of roof a house has would not be a good feature for our model. The two other features are numerical features where the numerical value of the feature takes on some important. This is not the case for the type of roof, as the numerical value simply represents a category.

   (d) The training complexity is $O(N * D)$, where N is the number of training samples and D is the number of features. We will have to go through all N training samples to find the nearest neighbor. We will also have to go through all D features to compute the distance between the test sample and the training sample.

   (e) The curse of dimensionality is a concept where once our training data has many many features, the distance between the nearest and the farthest neighbors becomes very similar. As the number of dimensions increases, the volume of the

space increases exponentially. This becomes a problem because our nearest neighbors are not going to be very predictive of the test sample (s).

6. (a) True
   (b) immediately before the last sigmoid activation function
   (c) $2K^2$
   (d) True
   (e) False

3 Python Programming Questions

- **Question 1:**
- **Question 2:**
- **Question 3:**
- **Question 4:**
- **Question 6:**