CS559: Machine Learning Name: Aidan Fischer Due: March 14, 2022

Assignment 3

Aidan Fischer

I pledge my honor that I have abided by the Stevens Honor System.

Homework assignments will be done individually: each student must hand in their own answers. Use of partial or entire solutions obtained from others or online is strictly prohibited. Electronic submission on Canvas is mandatory.

Submission Instructions You shall submit a zip file named Assignment3_LastName_FirstName.zip which contains:

- python files (.ipynb or .py) including all the code, comments, plots, and result analysis. You need to provide detailed comments in English.
- pdf file including (a) solutions for questions 1 and 2, (b) descriptions of observations for questions 3,4,5.

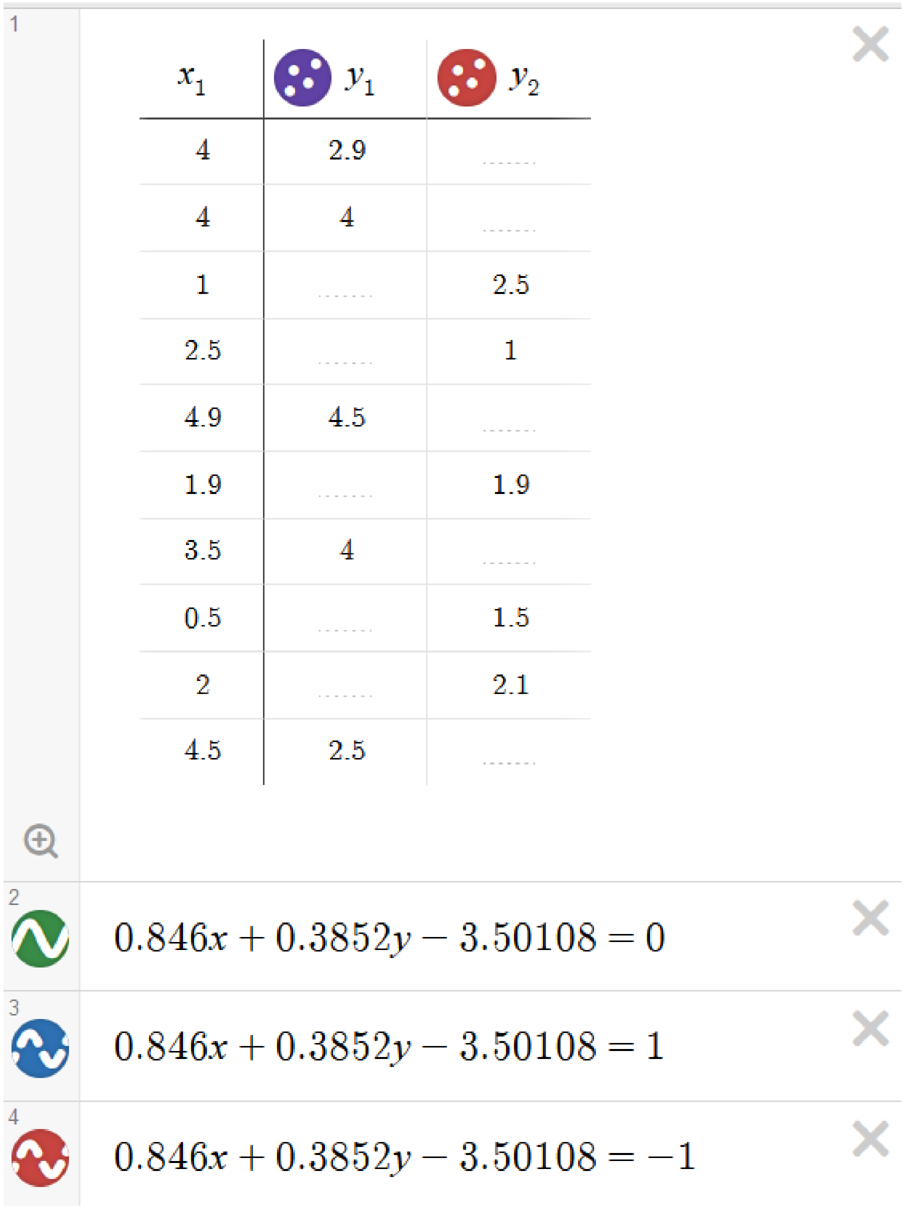
Table 1: Data set for Question 1

data	x_{i1}	x_{i2}	y_i	α_i
$\mathbf{x}_1$	4	2.9	1	0.414
$\mathbf{x}_2$	4	4	1	0
$\mathbf{x}_3$	1	2.5	-1	0
$\mathbf{x}_4$	2.5	1	-1	0.018
$\mathbf{x}_5$	4.9	4.5	1	0
$\mathbf{x}_6$	1.9	1.9	-1	0
$\mathbf{x}_7$	3.5	4	1	0.018
$\mathbf{x}_8$	0.5	1.5	-1	0
$\mathbf{x}_9$	2	2.1	-1	0.414
$\mathbf{x}_{10}$	4.5	2.5	1	0

1. **Support Vector Machines** (20 points) Given 10 points in Table 1, along with their classes and their Lagranian multipliers (α_i), answer the following questions:

   (a) (7 pts) What is the equation of the SVM hyperplane $h(x)$? Draw the hyperplane with the 10 points.
   The hyperplane is at $\mathbf{w}^T\mathbf{x} + b = 0$
   To figure this out, we need 2 pieces of information, the solutions for $\mathbf{w}$ and b.
   For SVM,
   $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$
   For this particular this, this gives
   $\mathbf{w} = 0.414 * 1 * (4, 2.9) + 0.018 * -1 * (2.5, 1) + 0.018 * 1 * (3.5, 4) + 0.414 * -1 * (2, 2.1)$
   $\mathbf{w} = (0.846, 0.3852)$
   To get b, we choose a support vector, and solve
   $\alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1] = 0$
   for b, where -1 is supposed to be $(1 - \xi_i)$, but estimating b sets ξ_i to 0

Selecting point $\mathbf{x}_i$,
$\alpha_i[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1] = 0$
$0.414[1((0.846, 0.3852)^T(4, 2.9) + b) - 1] = 0$
$0.414[1(4.50108 + b) - 1] = 0$
$0.414[3.50108 + b] = 0$
$3.50108 + b = 0$
$b = -3.50108$
Now, we have the hyperplane:
$(0.846, 0.3852)^T\mathbf{x} - 3.50108 = 0$
To draw the hyperplane, I placed the table of datapoints into Desmos, an online graphing calculator, as well as the hyperplane and margin functions.
Here is a picture of the graphing inputs.



1

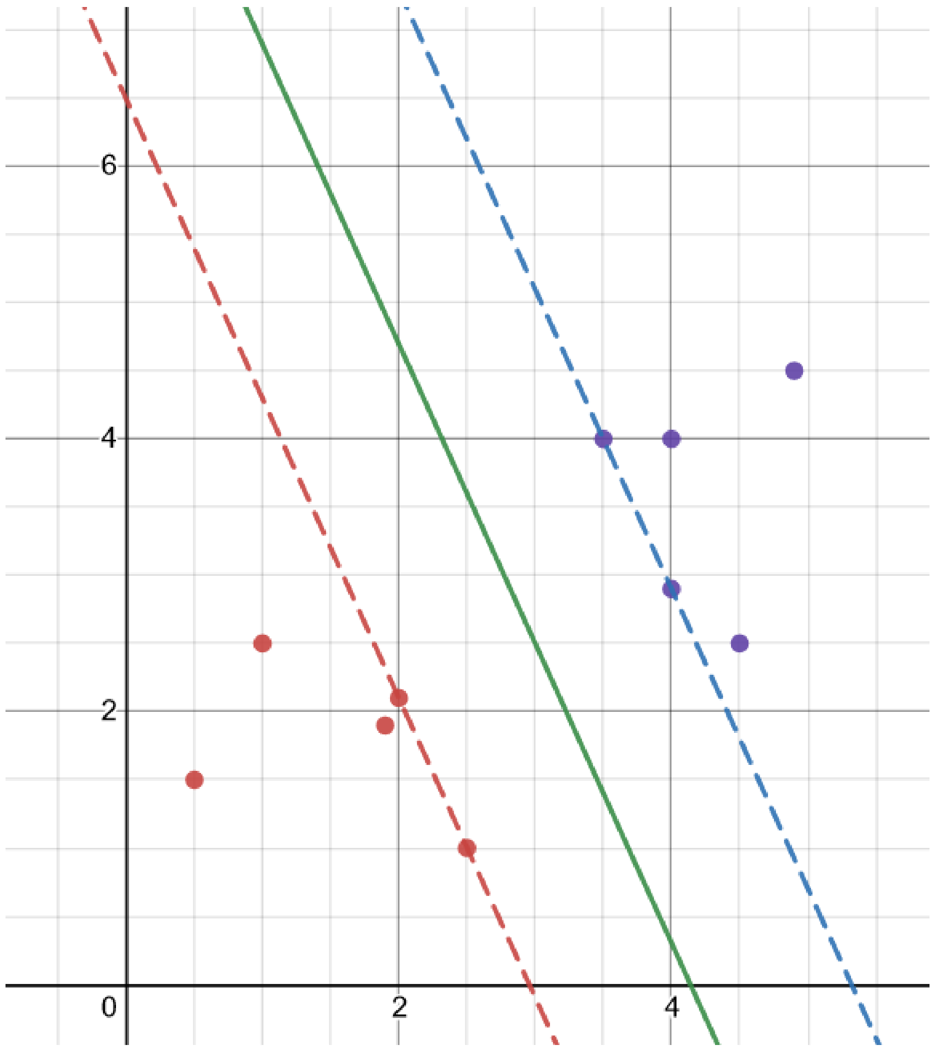
x_1	y_1	y_2
4	2.9	
4	4	
1		2.5
2.5		1
4.9	4.5	
1.9		1.9
3.5	4	
0.5		1.5
2		2.1
4.5	2.5	

2 $0.846x + 0.3852y - 3.50108 = 0$

3 $0.846x + 0.3852y - 3.50108 = 1$

4 $0.846x + 0.3852y - 3.50108 = -1$

Here is the graph output. Points classified as 1 are blue, points classified as -1 are red. The hyperplane is green, the class 1 margin is blue dotted, and the class -1 margin is red dotted.



(b) (8 pts) What is the distance of x_6 from the hyperplane? Write down the process of how the distance is calculated. Is it within the margin of the classifier?

Distance of a point to the hyperplane is given by

$d = \frac{1}{||\mathbf{w}||}(\mathbf{w}^T\mathbf{x}_i + b)$

Plugging in x_6, $\mathbf{w}$, and b, we obtain

$d = \frac{1}{||(0.846, 0.3852)||}((0.846, 0.3852)^T(1.9, 1.9) - 3.50108)$

$= \frac{1}{0.929567}((0.846, 0.3852)^T(1.9, 1.9) - 3.50108)$

$= \frac{1}{0.929567}(-1.1618)$

$= -1.249829$

Since this is less than -1, this point is outside the margin.

(c) (5 pts) Classify the point $z = (3,3)^\top$ using $h(x)$ from above.

Classifying z, we need to take

$\mathbf{w}^T z + b$

$= (0.846, 0.3852)^T(3,3) - 3.50108$

$= 0.19252$

This is positive, so it is classified as 1, however it is very near the boundary and is inside the margin.

2. **Support Vector Machines** (20 points) The SVM loss function with slack variables can be viewed as:

$$\min_{\mathbf{w},b} \frac{||\mathbf{w}||^2}{2} + \gamma \sum_{i=1}^{N} \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{Hinge loss}} \tag{1}$$

The hinge loss provides a way of dealing with datasets that are not separable.

(a) (8 pts)Argue that $l = \max(0, 1 - y\mathbf{w}^\top \mathbf{x})$ is convex as a function of $\mathbf{w}$.

A function is convex on an interval if its second derivative is greater than or equal to 0 for that entire interval.

We have two intervals with this function, one where $1-y\mathbf{w}^T\mathbf{x} > 0$, and the other where $1-y\mathbf{w}^T\mathbf{x} \leq 0$. There are only two intervals because this function is linear in $\mathbf{w}$

It is easy to see that the second derivative of l with respect to w is 0 for the entirety of the second interval, since if $1 - y\mathbf{w}^T\mathbf{x} \leq 0$, then $max(0, 1 - y\mathbf{w}^T\mathbf{x}) = 0$. Thus, for this interval, l is convex.

For the other interval, $1 - y\mathbf{w}^T\mathbf{x} > 0$, and thus l = $1 - y\mathbf{w}^T\mathbf{x}$.

The first derivative of l with respect to w in this interval is $-y\mathbf{x}$.

Thus, the second derivative is 0, and the function is convex on this interval as well.

Since l is convex over two intervals that cover it's entire domain, l itself is convex.

(b) (5 pts) Suppose that for some $\mathbf{w}$ we have a correct prediction of f with $\mathbf{x}_i$, i.e. $f(\mathbf{x}_i) = \text{sgn}(\mathbf{w}^\top \mathbf{x}_i)$. For binary classifications ($y_i = +1/-1$), what range of values can the hinge loss, l, take on this correctly classified example? Points which are classified correctly and which have non-zero hinge loss are referred to as margin mistakes.

Hinge loss is based on $f(x) = \mathbf{w}^T\mathbf{x}$, not on the sign of this function. The sign is the prediction, but hinge loss is based on the continuous value of the predictor function.

Hinge loss is $max(0, 1 - yf(\mathbf{x})$

For this case, since $y = sgn(f(\mathbf{x}))$, $yf(\mathbf{x}) = |f(\mathbf{x})|$

Thus, $l = max(0, 1 - |f(\mathbf{x})|)$

This only takes a value greater than 0 if $1 - |f(\mathbf{x})| > 0$

$\implies |f(\mathbf{x})| < 1$

Thus, $|f(\mathbf{x})|$ has a range of [0,1)

Thus, l can take on the range [0,1], since l will be zero if $|f(\mathbf{x})| \geq 1$

(c) (7 pts) Let $M(\mathbf{w})$ be the number of mistakes made by $\mathbf{w}$ on our dataset (in terms of classification loss). Show that:

$$\frac{1}{n} M(\mathbf{w}) \leq \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i)$$

In other words, the average hinge loss on our dataset is an upper bound on the average number of mistakes we make on our dataset.

First, I will argue that, for all mistakes, $\max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i) \geq 1$

In lieu of part b, but the opposite argument, if the classifier makes a mistake, that implies that $sgn(\mathbf{w}^\top\mathbf{x}_i)) = -y_i$, thus $y_i\mathbf{w}^\top\mathbf{x}_i = -|\mathbf{w}^\top\mathbf{x}_i|$

Thus, $\max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i) = \max(0, 1 + |\mathbf{w}^\top\mathbf{x}_i|)$

Since $|\mathbf{w}^\top\mathbf{x}_i| \geq 0$, $\max(0, 1 + |\mathbf{w}^\top\mathbf{x}_i|) = 1 + |\mathbf{w}^\top\mathbf{x}_i| \geq 1$

From the previous problem, all non-mistakes have hinge loss $0 \leq \text{l} \leq 1$

Now,

$\frac{1}{n}M(\mathbf{w}) \leq \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i)$

$\implies M(\mathbf{w}) \leq \sum_{i=1}^{n} \max(0, 1 - y_i\mathbf{w}^\top\mathbf{x}_i)$

Now, all non-mistakes strictly contribute 0 to $M(\mathbf{x})$, while they contribute a number $0 \leq l \leq 1$ to total hinge loss. All mistakes strictly contribute 1 to $M(\mathbf{x})$, while they contribute a number $l \geq 1$ to total hinge loss.

Thus, $M(\mathbf{x}_i) \leq max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i) \; \forall \; \mathbf{x}_i$, thus the total M is less than or equal to total hinge loss, and thus the average number of mistakes is less than average hinge loss.

3. **Decision Trees** (20 points) Implement a Decision Tree model for the Titanic data set (only use the train file).

- (2 pts) Explain how you preprocess the features.

  Step 1: Manual processing:

  Deleted from the data two features, passengerId and ticketNumber, because id is completely unrelated to the data, and is unique, and because ticket numbers are random, and also unique to every passenger. For the same reasons, I removed the name column as well.

  I decided to keep ticket fare. Despite it being seemingly unrelated to result, I kept it because it is a unique way of combining previous features (i.e. ticket fare could depend on age, pclass, sex, cabin, and where the person embarked).

  I decided to get rid of the cabin feature. Many individuals did not have one assigned, others had multiple. I just did not see it performing well as a feature in a decision tree. Had this not been the case, I would've bucketed them by letter (C2 -¿ C, D345 -¿ D, etc).

  So, I kept the Survived (which is the actual result column), Pclass, Sex, Age, SibSp, Parch, Fare, and Embarked columns.

  Step 2: Programmatic processing: Inf Load examples into an array of dictionaries representing the example, where each key is the feature's name. This is as part of an Example class, which stores the actual result data ("survived" column) and the actual training columns seperately.

  Since decision trees act on discrete data points, I thought placing fare, which is a continuous variable, and age, which isn't continuous but has a large range, into buckets was a good idea. I.e., I split age into buckets of 10 (0-9, 10-19, etc), and fare into buckets of \$25 (0-24, 25-50, etc).

  Some of ages are not assigned, however majority have ages as opposed to the cabin feature, and only ever have at most 1 age, so I decided to keep age, and assign "" (Empty String) as it's own bucket. I am unsure if some fares are missing, but I do the same thing here. All non-bucketed features automatically consider empty data as its own value.

  This way we avoid having a tree that just classifies a solution based on ticket prices and ages, which doesn't give us any information and would almost certainly not work on untrained data. However, removing these features would not be a good idea, as one's age may very well have an influence on whether they had survived, and I already justified keeping fare.

  I also tested the method for real-valued variables given in the slide, i.e. thresholding, but I'm not sure I implemented it correctly.

- (2 pts) Divide the train file into a training set and a test set.

  Splits the data into 80% train, 20% test.

- (8 pts) Build a tree on the training data and evaluate the classification performance on the test data.

  Age and Fare are the attributes that are thresholded/bucketed.

  Done using Thresholded and Bucketed measures, using Gini index and information gain algorithms.

- (4 pts) Try both Gini index and Information Gain as the attribute selection in building the decision tree; Compare their results on the test set.

  Accuracy on test data:

  Bucketed Inf Gain: 73.03%

  Bucketed Gini Index: 73.03%

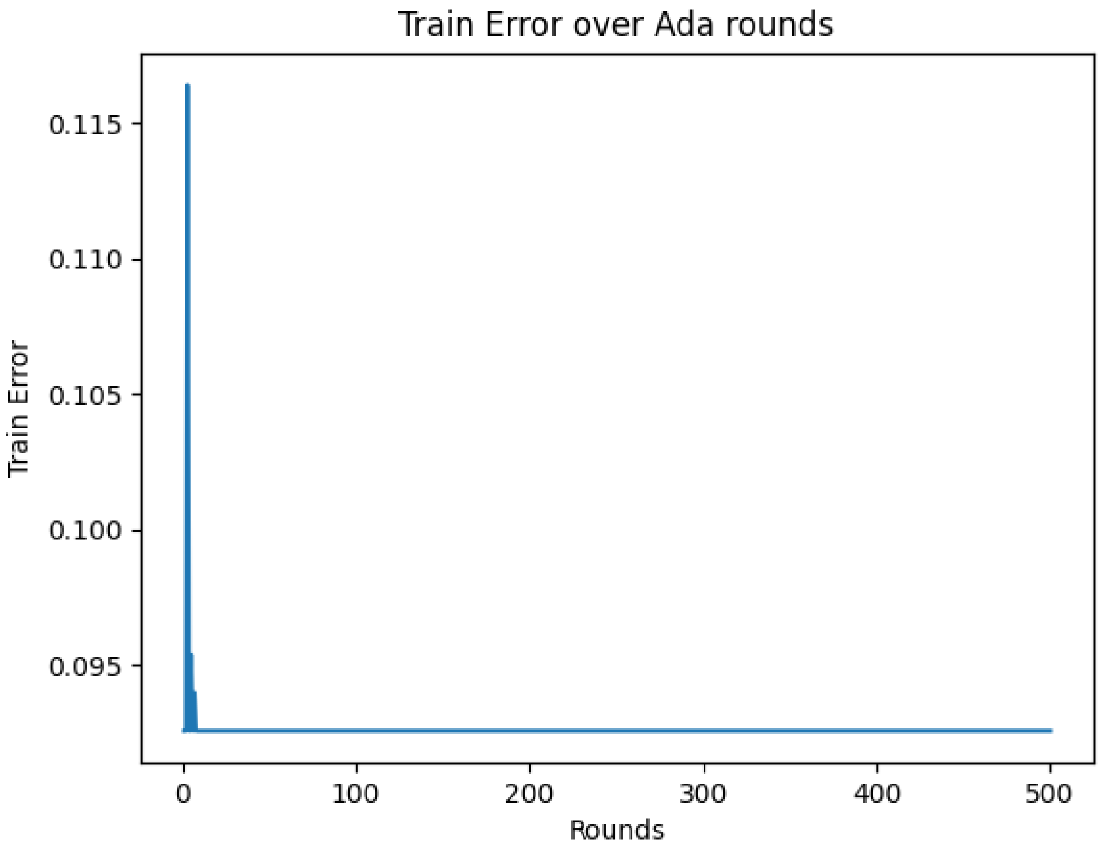
  Thresholded Inf Gain: 71.91%

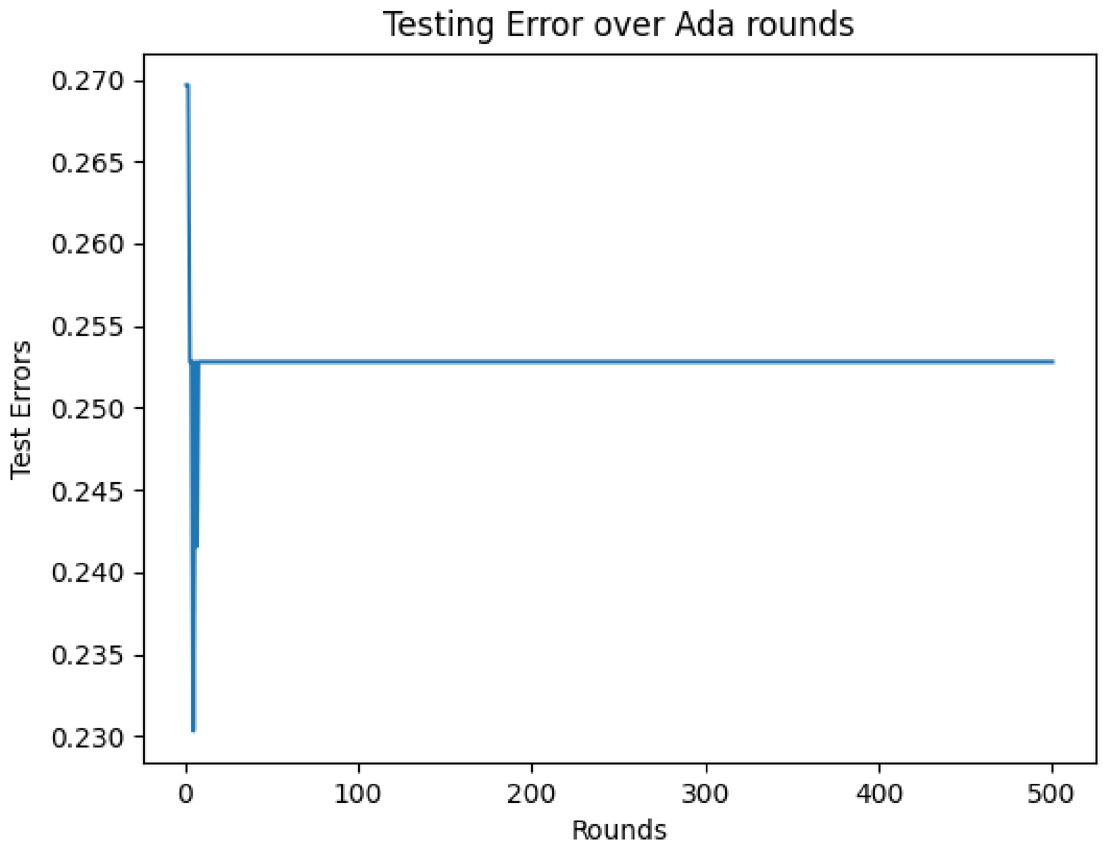
  Thresholded Gini Index: 71.91%

- (2 pts) Report your best accuracy on the test data set.
  73.03%
- (2 pts) Give a brief description of your observations.
  Interestingly, the two algorithms gave the same model/train accuracy for each of the continuous variable management methods. Both Gini and Inf had a model test accuracy of 73% for bucketed real variables, and 72% for thresholded real variables. However, I know that the two algorithms generated different decision trees, because I implemented a method of printing the trees to a file. The resulting trees are in the Results folder of my submission. My guess is that the two algorithms generated trees that were different but essentially equivalent, at least with respect to the test and training sets. However, it is possible that while accuracy is the same, the ones it misses are different. I did not check this. While I did not include this data in this report, the models' accuracy on the training set also tell an interesting story. Bucketed algorithms had a training accuracy of $\sim 90\%$, while thresholded algorithms had an accuracy $\sim 70\%$, putting the test and train accuracies far closer.

4. **Boosting** (20 points) Implement AdaBoost for the Titanic data set. You can use package/tools to implement your decision tree classifiers. The fit function of DecisionTreeClassifier in sklearn has a parameter: sample weight, which you can use to weigh training examples differently during various rounds of AdaBoost.

   - (10 pts) Implement the AdaBoost algorithm.
     I did end up using the sklearn package. I create a class that stores the current weights, a list of trained trees, a list of alphas, an OrdinalEncoder, which converts non-numeric features to a form usable by sklearn, the set of training data passed through the encoder, and the set of actual results for each of the training set. My class defines the necessary adaboost functions, which include a function that predicts all training inputs through the last trained tree, calculation of e and alpha on the last trained tree, and a function that performs the adaboost iteration that updates the weights.
   - (4 pts) Plot the train and test errors as a function of the number of rounds from 1 through 500.


Train Error over Ada rounds
Train Error
0.115
0.110
0.105
0.100
0.095
Rounds
0
100
200
300
400
500



- (3 pts) Report your best accuracy on the test data set.
  Lowest Training Error: 0.09256661991584852
  Lowest Testing Error: 0.2303370786516854
  Last Training Error: 0.09256661991584852
  Last Testing Error: 0.25280898876404495
  Best accuracy on test dataset, near the early rounds, is 23.03%
- (3 pts) Give a brief description of your observations.
  The training error quickly plateaued at 9.25%, while the testing error quickly dropped to its lowest error, then jumped to around 25.28% and also plateaued at that level.

5. **Neural Networks** (20 points) Apply a Neural Network (NN) model to predict a handwritten digit images into 0 to 9. The pickled file represents a tuple of 3 lists : the training set, the validation set and the test set. Each of the three lists is a pair formed from a list of images and a list of class labels for each of the images. An image is represented as numpy 1-dimensional array of 784 (28 x 28) float values between 0 and 1 (0 stands for black, 1 for white). The labels are numbers between 0 and 9 indicating which digit the image represents. The code block below shows how to load the dataset.

```
import cPickle, gzip, numpy

# Load the dataset
f = gzip.open('mnist.pkl.gz', 'rb')
train_set, valid_set, test_set = cPickle.load(f)
f.close()https://www.overleaf.com/project/6224d5d6caa82d8e45bc7a6a
```

- (6 pts) Build a neural network using the training data and predict the labels of data samples in the test set. You can use packages for neural networks. If you write a neural network from scratch with forward pass and back-propagation, you will get extra credits.

  I used sklearn's neural network class. I set it up to use stochastic gradient descent as the learning algorithm, and chose the logistic activation function.

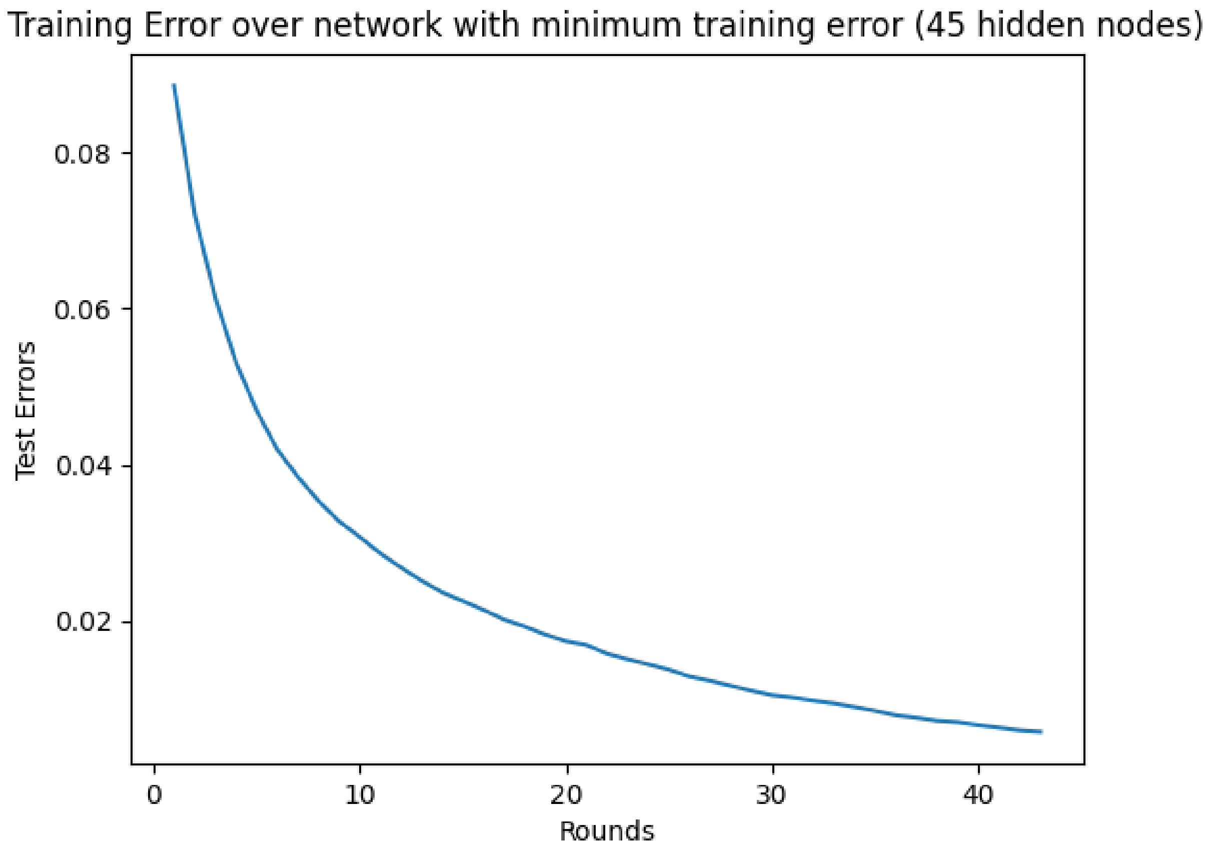
  In order to keep track of the errors over the training iterations, as well as to have the ability to manually apply early stopping through the provided validation set, I used the partial_fit function, which runs a single iteration of the training, as opposed to the fit function, which runs until completion.

- (3 pts) Tune your model hyper-parameters (number of hidden neurons in your hidden layer) on the validation set.

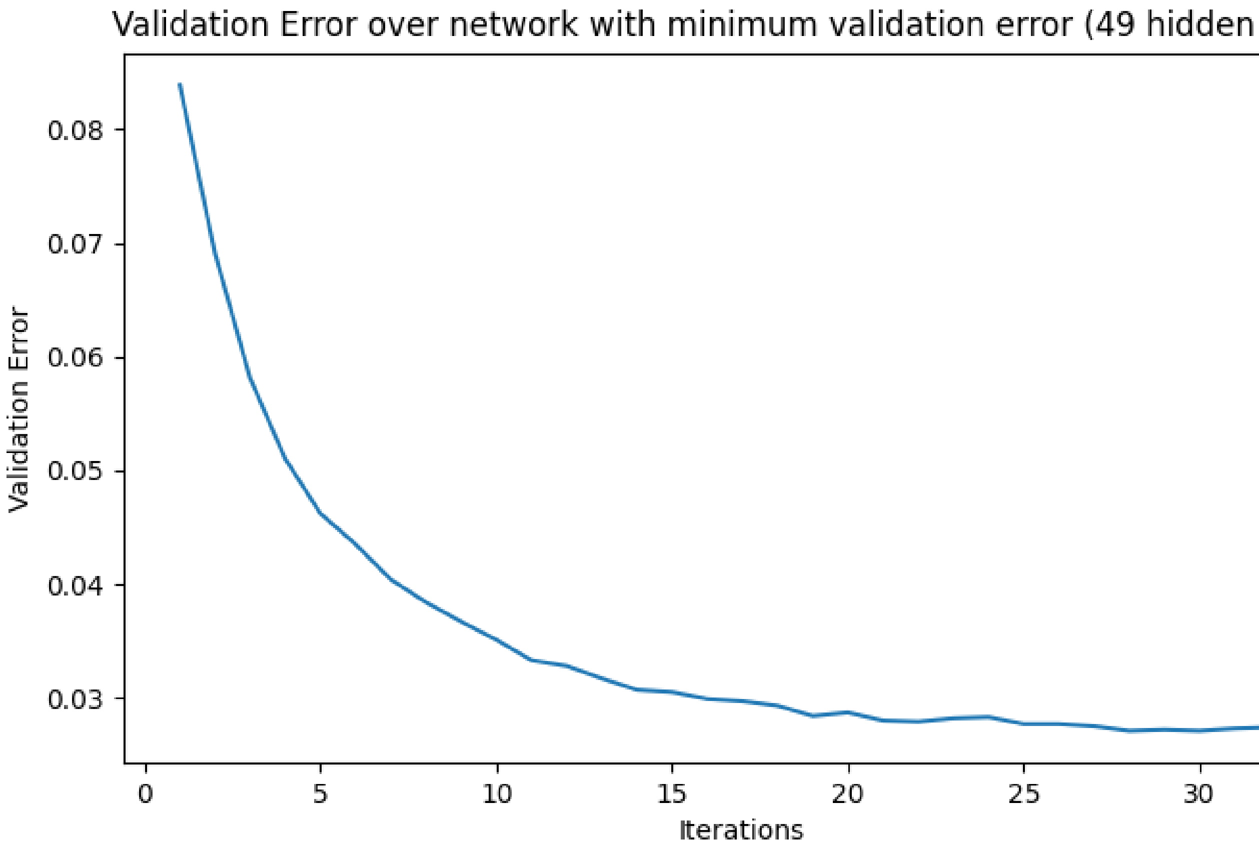
  I looped through 1 to 30 hidden layer nodes and trained each of the resulting networks.

- (5 pts) Plot the train, validation, and test errors as a function of the epoches.

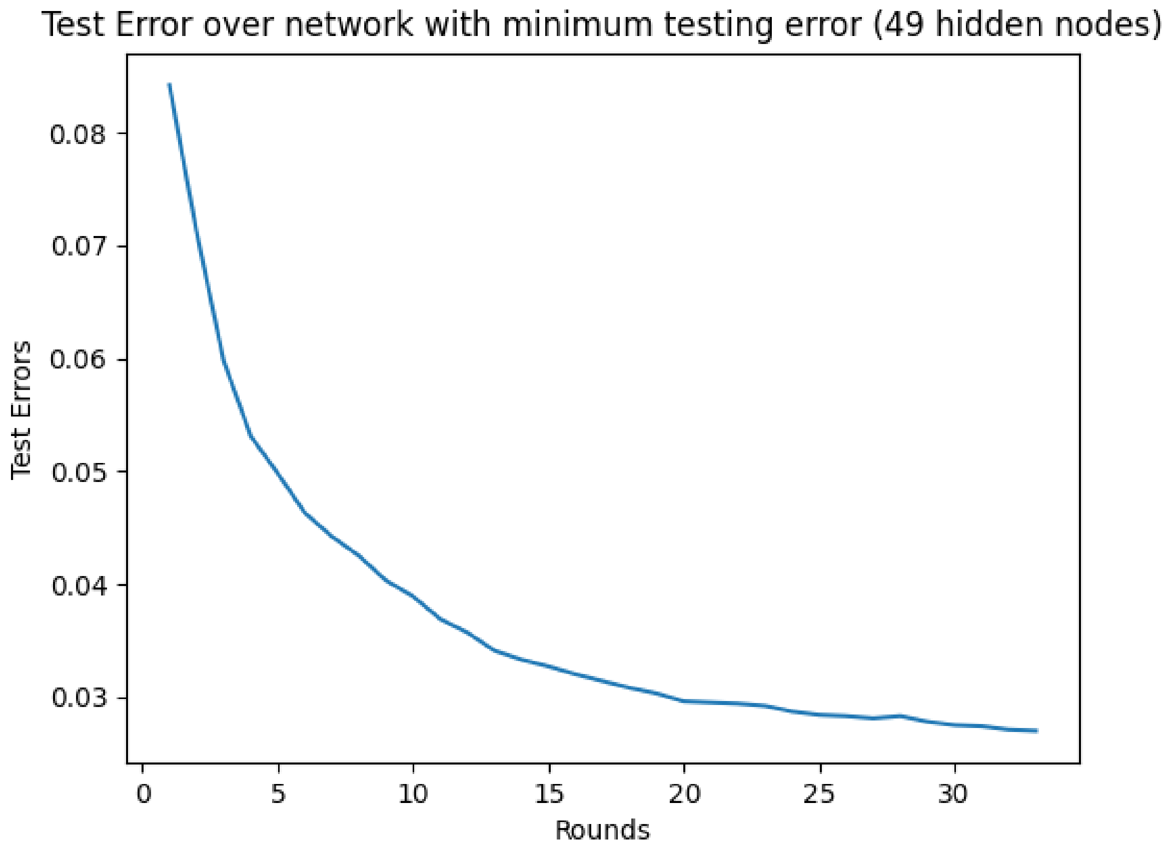
  Train error over iterations from best model wrt Train error:



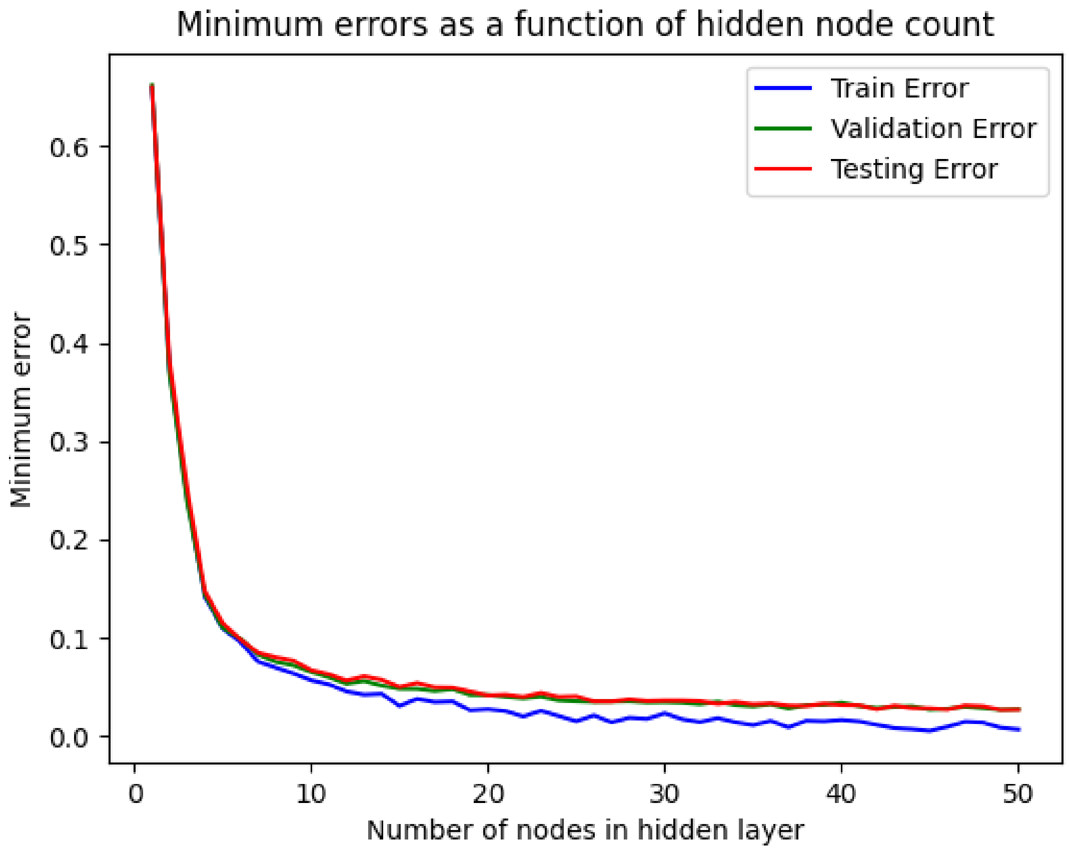
Validation error over iterations from best model wrt Validation error:



Test error over iterations from best model wrt Test error



All errors graphed as a function of number of hidden nodes



- (2 pts) Report the best accuracy on the validation and test data sets.

  Best Validation Accuracy: 97% Best Test Accuracy: 97%

- (2 pts) Apply early stopping using the validation set to avoid overfitting.

  I kept track of whether the training steps were actually improving training/validation error by more than a tolerance level of 0.0001. If either training or validation error failed to improve by that level or more (whether it was improving by a smaller amount, or actually getting worse) for more than 5 training iterations, I halted the training for that model.

- (2 pts) Give a brief description of your observations.

  I noticed the minimum error decreased the more hidden nodes we included. However, training on a higher number of nodes resulted in more training iterations. Finally, the actual improvement seen from including more hidden nodes sharply fell off after around 10-15 hidden nodes, as seen in the minimum errors as a function of hidden nodes graph above. A small number of hidden nodes, where the number of hidden nodes is less than the number of output possibilities, i.e.e between 1 and 10 hidden nodes, the errors were very high, between 10 and 60%