HOMEWORK 2

Matt Myers
908-464-4252
GitHub

Instructions: Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file to Canvas. Please wrap your code and upload to a public GitHub repo, then attach the link below the instructions so that we can access it. You can choose any programming language (i.e. python, R, or MATLAB), as long as you implement the algorithm from scratch (e.g. do not use sklearn on questions 1 to 7 in section 2). Please check Piazza for updates about the homework.

1 A Simplified Decision Tree

You are to implement a decision-tree learner for classification. To simplify your work, this will not be a general purpose decision tree. Instead, your program can assume that

- each item has two continuous features $\mathbf{x} \in \mathbb{R}^2$
- the class label is binary and encoded as $y \in \{0, 1\}$
- data files are in plaintext with one labeled item per line, separated by whitespace:

$$x_{11} \quad x_{12} \quad y_1$$
$$\ldots$$
$$x_{n1} \quad x_{n2} \quad y_n$$

Your program should implement a decision tree learner according to the following guidelines:

- Candidate splits (j, c) for numeric features should use a threshold c in feature dimension j in the form of $x_{\cdot j} \geq c$.
- c should be on values of that dimension present in the training data; i.e. the threshold is on training points, not in between training points. You may enumerate all features, and for each feature, use all possible values for that dimension.
- You may skip those candidate splits with zero split information (i.e. the entropy of the split), and continue the enumeration.
- The left branch of such a split is the "then" branch, and the right branch is "else".
- Splits should be chosen using information gain ratio. If there is a tie you may break it arbitrarily.
- The stopping criteria (for making a node into a leaf) are that
  - the node is empty, or
  - all splits have zero gain ratio (if the entropy of the split is non-zero), or
  - the entropy of any candidates split is zero
- To simplify, whenever there is no majority class in a leaf, let it predict $y = 1$.

2 Questions

1. (Our algorithm stops at pure labels) [10 pts] If a node is not empty but contains training items with the same label, why is it guaranteed to become a leaf? Explain. You may assume that the feature values of these items are not all the same.

   If the node is not empty but contains training items with the same label the entropy of all splits is zero. Due to our stopping criteria, if the entropy of any candidates split is zero we should stop and create a leaf. This makes sense because they all have the same value of one or zero, therefore the data is perfectly represented.

2. (Our algorithm is greedy) [10 pts] Handcraft a small training set where both classes are present but the algorithm refuses to split; instead it makes the root a leaf and stop; Importantly, if we were to manually force a split, the algorithm will happily continue splitting the data set further and produce a deeper tree with zero training error. You should (1) plot your training set, (2) explain why. Hint: you don't need more than a handful of items.

   One such training set would be as follows:

x_{n_1}	x_{n_2}	y_n
1	2	0
1	2	1
3	4	0
3	4	1

$\Longrightarrow$

x_{n_1}	x_{n_2}	y_n
1	2	0
3	4	1

$+$

x_{n_1}	x_{n_2}	y_n
1	2	1
3	4	0

   To further explain why this works, I will explain how our algorithm works. With this training set our algorithm will look at the x_{n_1} and x_{n_2} to find that for both the first and second column they are the same pair (x_{n_1}, x_{n_2}) but have different y_n values. This can't be because we would have a pair that would lead to both zero and one. For example this would be like saying when it is hot and rainy you never play tennis. Then next time it is hot and rainy you have to play tennis. The algorithm will not split at this point. If we force a split as shown above however our algorithm will have no problem going through the rest of the data.

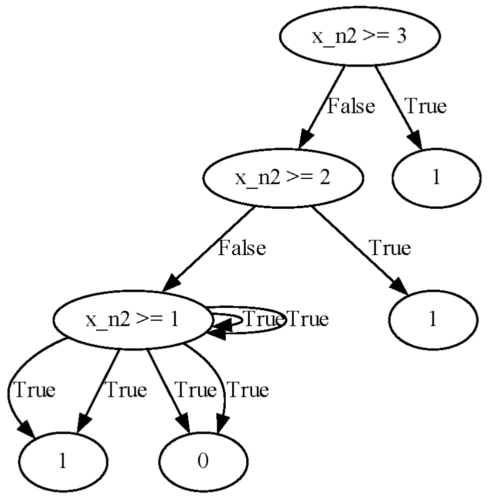
3. (Information gain ratio exercise) [10 pts] Use the training set Druns.txt. For the root node, list all candidate cuts and their information gain ratio. If the entropy of the candidate split is zero, please list its mutual information (i.e. information gain). Hint: to get $\log_2(x)$ when your programming language may be using a different base, use `log(x)/log(2)`. Also, please follow the split rule in the first section.

Feature: x_n1
Split Value: 0.0
Gain Ratio: 0.10051807676021828
InfoGain: 0.044177391867261 33
Entropy: 0.4394969869215134
Feature: x_n1
Split Value: 0.0
Gain Ratio: 0.11124029586339801
InfoGain: 0.10519553207004628
Entropy: 0.9456603046006401
Feature: x_n1
Split Value: 0.0
Gain Ratio: 0.23609960614360798
InfoGain: 0.19958702318968735
Entropy: 0.8453509366224364
Feature: x_n1
Split Value: 0.0
Gain Ratio: 0.055953759631263526
InfoGain: 0.03827452220629246

```
    Entropy: 0.6840384356390417
Feature: x_n1
    Split Value: 0.0
    Gain Ratio: 0.4301569161309807
    InfoGain: 0.18905266854301617
    Entropy: 0.4394969869215134
Feature: x_n2
    Split Value: -2
    Gain Ratio: 0.10051807676021828
    InfoGain: 0.044177391867261 33
    Entropy: 0.4394969869215134
Feature: x_n2
    Split Value: -1
    Gain Ratio: 0.055953759631263526
    InfoGain: 0.03827452220629246
    Entropy: 0.6840384356390417
Feature: x_n2
    Split Value: 5
    Gain Ratio: 0.23609960614360798
    InfoGain: 0.19958702318968735
    Entropy: 0.8453509366224364
Feature: x_n2
    Split Value: 6
    Gain Ratio: 0.055953759631263526
    InfoGain: 0.03827452220629246
    Entropy: 0.6840384356390417
Feature: x_n2
    Split Value: 7
    Gain Ratio: 0.4301569161309807
    InfoGain: 0.18905266854301617
    Entropy: 0.4394969869215134
```

4. (The king of interpretability) [10 pts] Decision tree is not the most accurate classifier in general. However, it persists. This is largely due to its rumored interpretability: a data scientist can easily explain a tree to a non-data scientist. Build a tree from D3leaves.txt. Then manually convert your tree to a set of logic rules. Show the tree[1] and the rules.

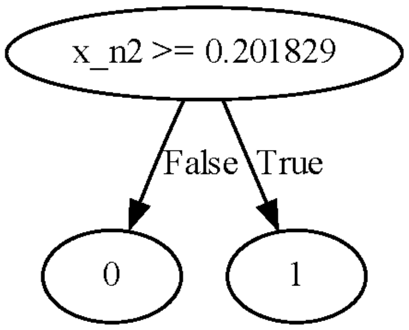
[1]When we say show the tree, we mean either the standard computer science tree view, or some crude plaintext representation of the tree – as long as you explain the format. When we say visualize the tree, we mean a plot in the 2D $\mathbf{x}$ space that shows how the tree will classify any points.



I realize that this is not correct and unfortunately no matter how much time I have spent on this I still cannot get it to work. I know that this is not quite right, I plan on going back and fixing this but I am now out of time. I know that the splits are supposed to be at $x_{n_1} > 1$ and then $x_{n_2} > 1$. I have spent countless hours working on this but I do have some of the charts looking good and I know with some more time I can fix this.

5. (Or is it?) [20 pts] For this question only, make sure you DO NOT VISUALIZE the data sets or plot your tree's decision boundary in the 2D $\mathbf{x}$ space. If your code does that, turn it off before proceeding. This is because you want to see your own reaction when trying to interpret a tree. You will get points no matter what your interpretation is. And we will ask you to visualize them in the next question anyway.

   - Build a decision tree on D1.txt. Show it to us in any format (e.g. could be a standard binary tree with nodes and arrows, and denote the rule at each leaf node; or as simple as plaintext output where each line represents a node with appropriate line number pointers to child nodes; whatever is convenient for you). Again, do not visualize the data set or the tree in the $\mathbf{x}$ input space. In real tasks you will not be able to visualize the whole high dimensional input space anyway, so we don't want you to "cheat" here.
   - Look at your tree in the above format (remember, you should not visualize the 2D dataset or your tree's decision boundary) and try to interpret the decision boundary in human understandable English.



The simplest way to understand the decision tree is the follows:

If the feature x_{n_2} is greater than or equal to 0.201829 than it is a 1 otherwise it is a 0. This is the simplest way to read a decision tree if the node is true then you follow the truth path otherwise you follow the false path.

- Build a decision tree on D2.txt. Show it to us.

  I created a decision tree for 'D2.txt' however it is much to large to see and I know exactly what is wrong with it. I however, do not have time to fix it right now but I plan on fixing it in the future. Regardless it will be included on the github page.

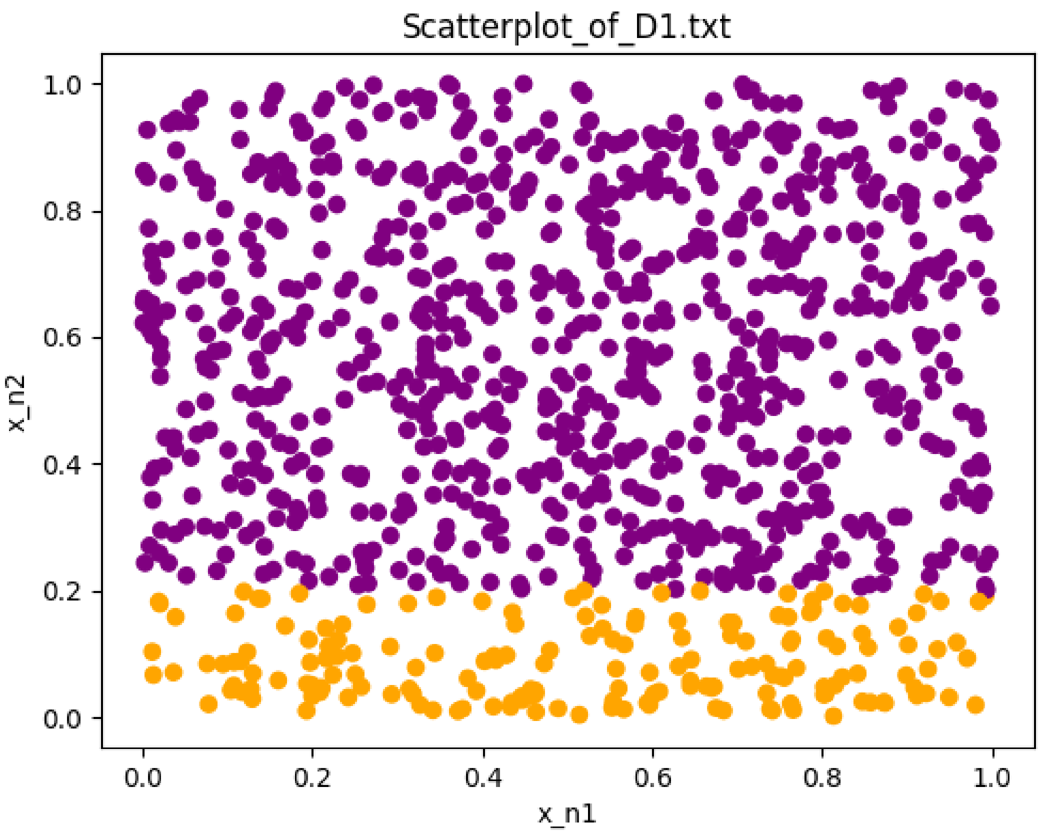
- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?

  The decision tree for D2 might not even be possible to understand without the visualization. This is because of 10,000 points there is a lot there. This data is overwhelming and would be near impossible to understand correlations without the decision tree. It is possible to interpret without a visualization but nearly impossible it would be extremely difficult.

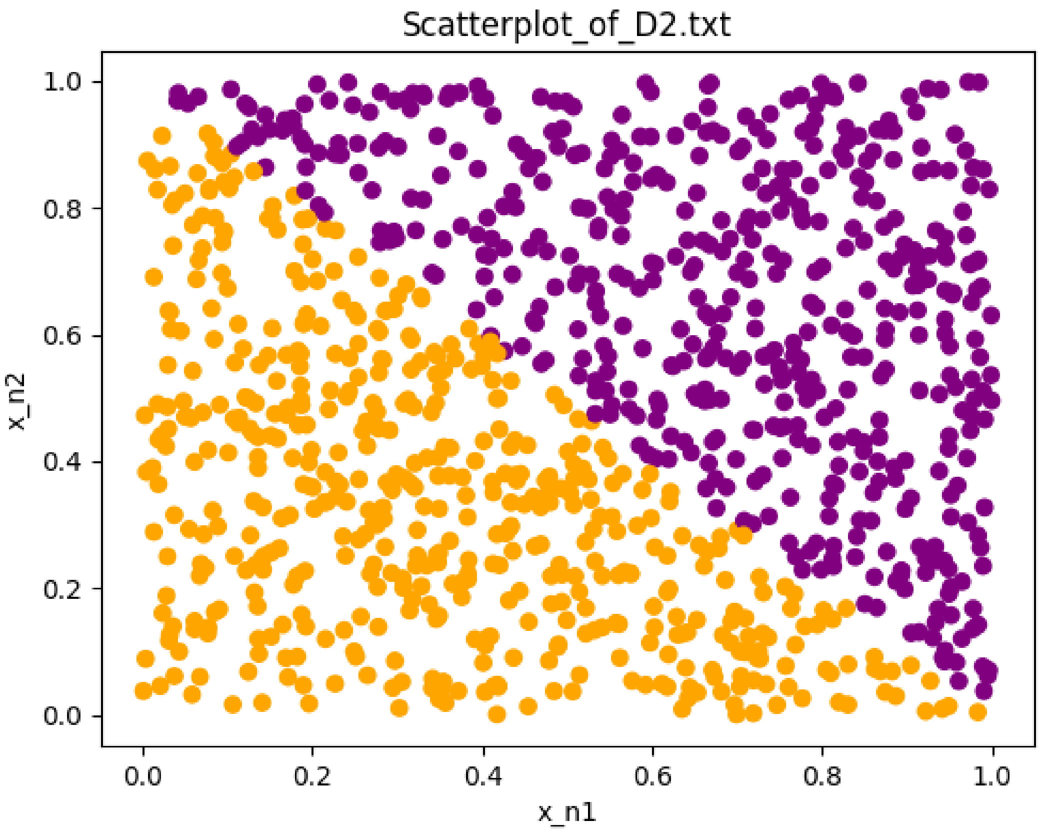
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

   - Produce a scatter plot of the data set.
   - Visualize your decision tree's decision boundary (or decision region, or some other ways to clearly visualize how your decision tree will make decisions in the feature space).

   Then discuss why the size of your decision trees on D1 and D2 differ. Relate this to the hypothesis space of our decision tree algorithm.



This is the scatterplot for the 'D1.txt' file. All of the purple dots are 1 and all of the orange dots are 0. Due to this coloring it is easy to see that while it is hard to predict anything about the data in x_{n_1} but in x_{n_2} we can see a clear speration around 0.2. This is why we see a perfect split of the data around 0.2 in the feature x_{n_2}.



This is the scatterplot for the 'D2.txt' file. All of the purple dots are 1 and all of the orange dots are 0. Due to this coloring it is easy to see that why it is so hard to predict the split. This is because it keeps spliting back and forth along the diagonal of this data. That is why we end up with so many splits. There is nearly no correlation between x_{n_1} and x_{n_2}.

7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

   - You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.

   ```
   # Creating random set
   df = data_raw['Dbig.txt']
   df = df.sample(frac=1)

   ```

   - Generate a sequence of five nested training sets $D_{32} \subset D_{128} \subset D_{512} \subset D_{2048} \subset D_{8192}$ from the candidate training set. The subscript n in D_n denotes training set size. The easiest way is to take the first n items from the (same) permutation above. This sequence simulates the real world situation where you obtain more and more training data.

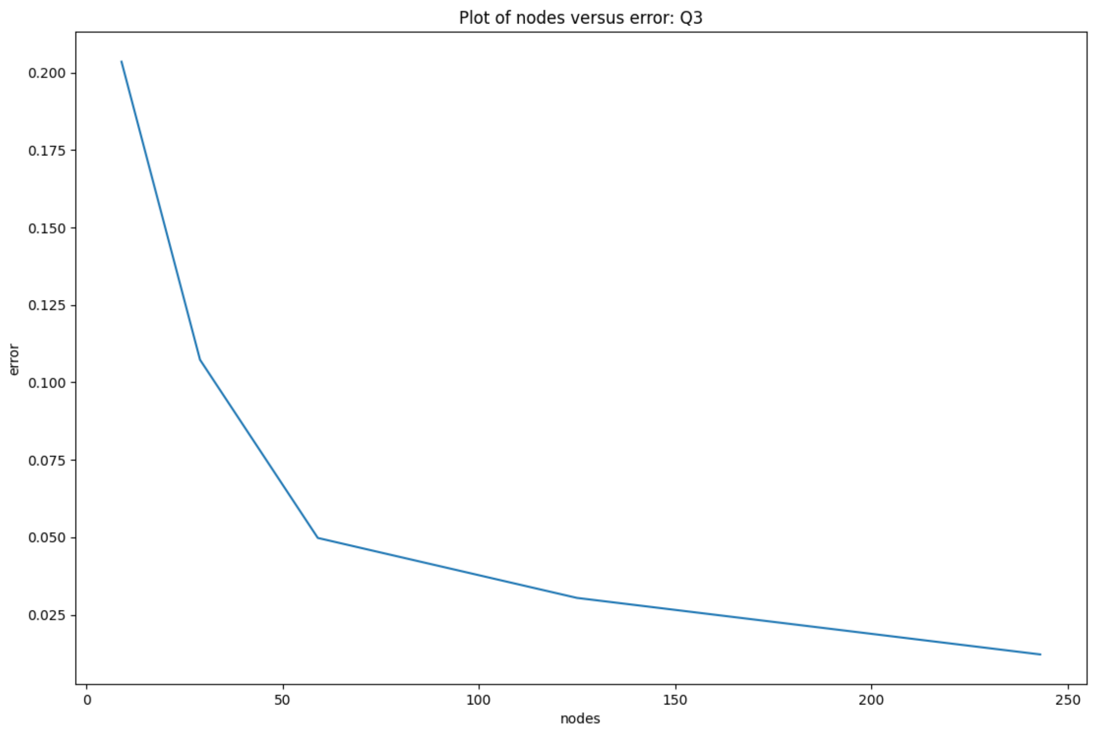
```
# Initializing the sets of data
df_32= df.iloc[:32]
df_128= df.iloc[:128]
df_512= df.iloc[:512]
df_2048= df.iloc[:2048]
df_8192= df.iloc[:8192]
df_test= df.iloc[8192:]
features=['x_n1','x_n2']
y = 'y_n'

```

- For each D_n above, train a decision tree. Measure its test set error err_n. Show three things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n. This is known as a learning curve (a single plot). (3) Visualize your decision trees' decision boundary (five plots).

3 sklearn [10 pts]

Learn to use sklearn (https://scikit-learn.org/stable/). Use sklearn.tree.DecisionTreeClassifier to produce trees for datasets $D_{32}, D_{128}, D_{512}, D_{2048}, D_{8192}$. Show two things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n.

dataset	Error	Number of Nodes
D_{32}	0.14103982300884957	15
D_{128}	0.08351769911504425	25
D_{512}	0.05309734513274336	55
D_{2048}	0.02765486725663717	115
D_{8192}	0.017146017699115043	225

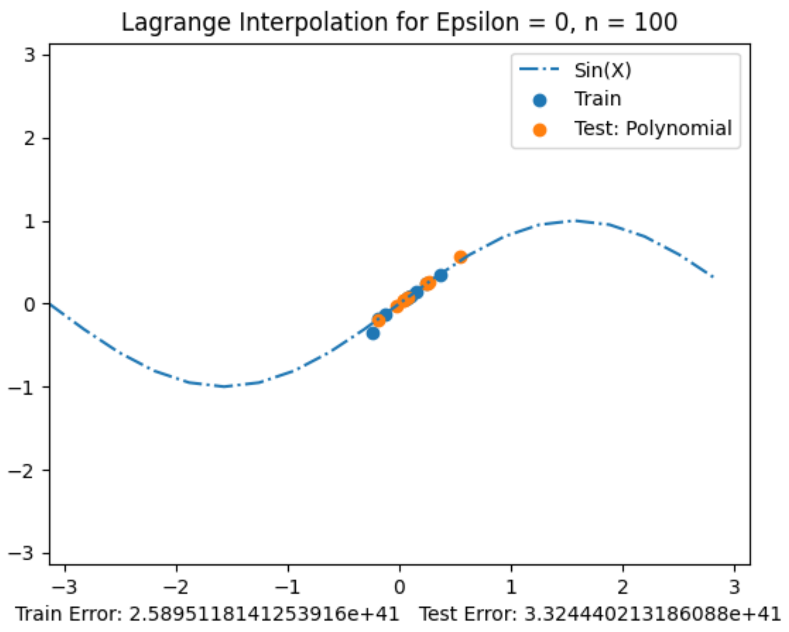


4 Lagrange Interpolation [10 pts]

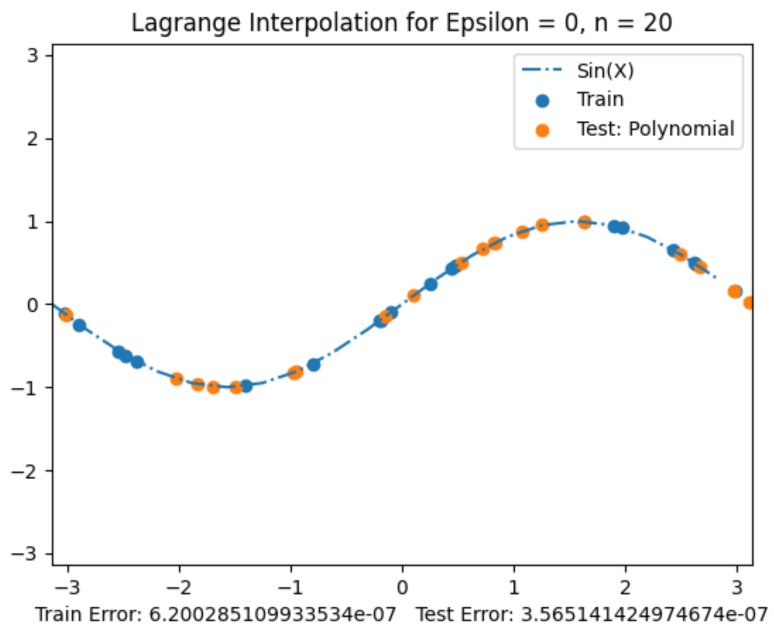
Fix some interval $[a, b]$ and sample $n = 100$ points x from this interval uniformly. Use these to build a training set consisting of n pairs (x, y) by setting function $y = sin(x)$.

Build a model f by using Lagrange interpolation, check more details in https://en.wikipedia.org/wiki/Lagrange_polynomial and https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.lagrange.html.

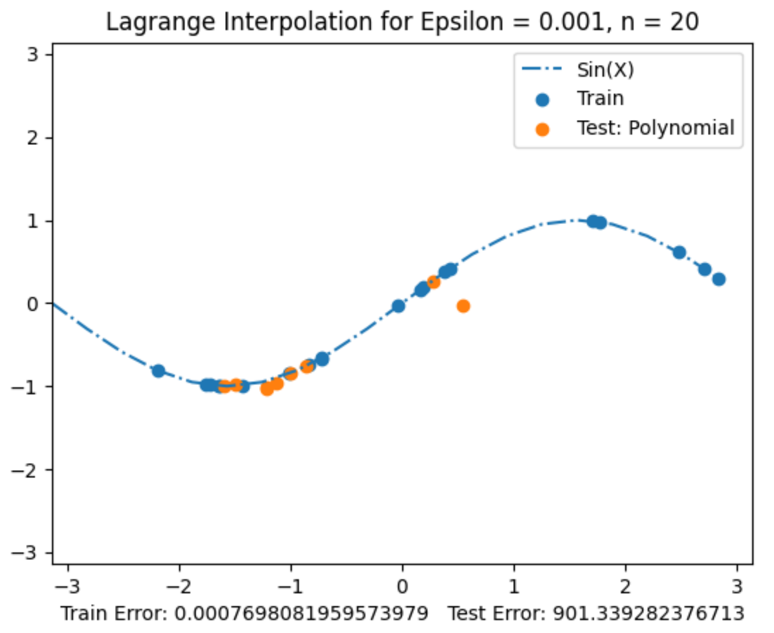
Generate a test set using the same distribution as your test set. Compute and report the resulting model's train and test error. What do you observe? Repeat the experiment with zero-mean Gaussian noise ϵ added to x. Vary the standard deviation for ϵ and report your findings.



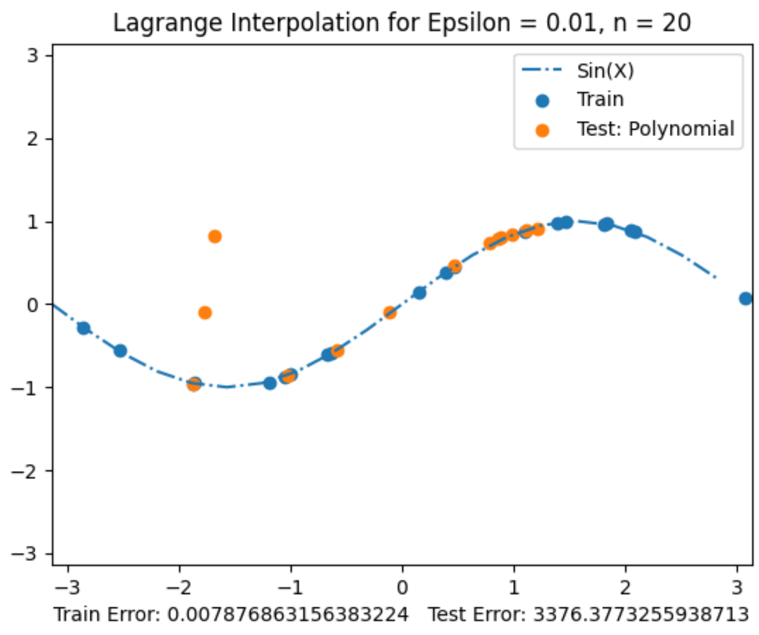
This is the first figure where $n = 100$ and $\epsilon = 0$. This is the model of the train and test sets one top of the actual data. As you can see it is not very accurate even though it might seem like it. The errors are so high because n is extremely large. As you will see in proceeding sketches a lower n will lead to a greater accuracy.



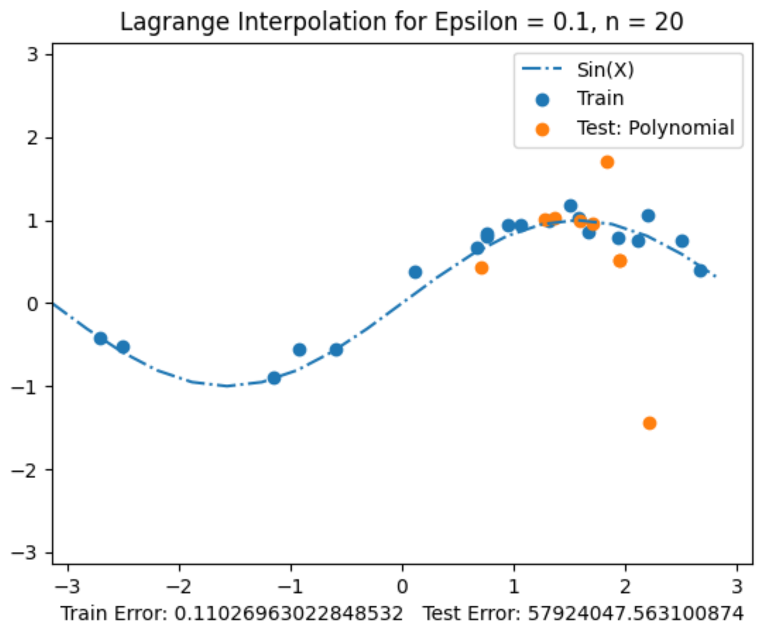
This is figure also has $\epsilon = 0$ but $n = 20$. As you can see the error is considerable lower (magnitude of 10^{48}). When $\epsilon = 0$ both the training and testing data is very accurate.



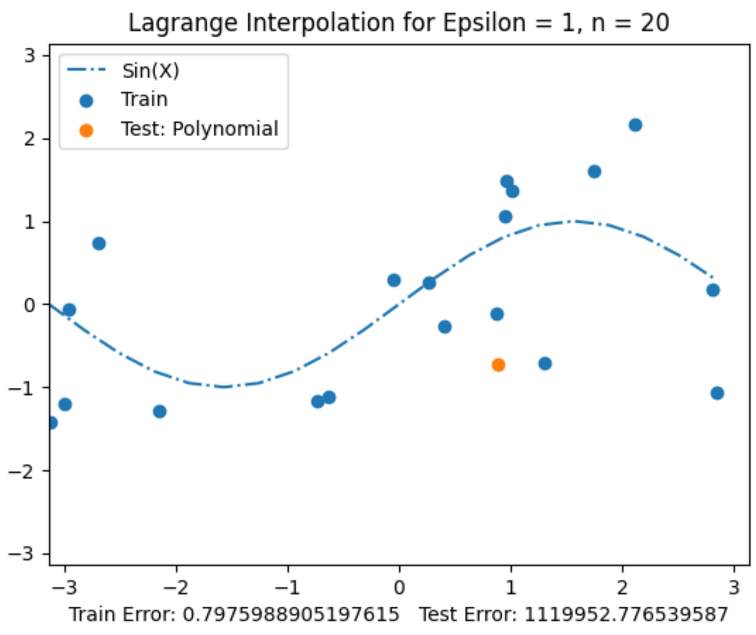
As ϵ begins to increase we will see the accuracy of both the training set and the testing set will go up. The training set will have much less error than the testing set as we would expect.



As ϵ continues to increase we will see the accuracy of both the training set and the testing set will go up. The training set will have much less error than the testing set as we would expect. This figure continues wtih the trend.



As you can see with the test points at this point the training error is quite bad. More error than we would ever want on something that we are using to predict. This shows how even small noise can greatly impact the result.



Finally when we have $\epsilon = 1$ we have way too much noise to be helpful at all. The training data is still relatively good which we would expect but the test data is useless at this point. Very interesting to see the negative correlation with noise as we would expect.

https://github.com/myersmt/Hw002-Comp760