HOMEWORK 2

Yichen Lin
908 531 9599

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.

link to code: https://github.com/whocansavemyhair/test.git

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:

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

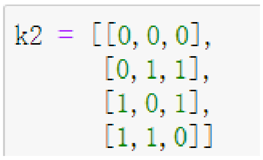
Your program should implement a decision tree learner according to the following guidelines:

- 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.
   Answer: Yes, it will become a leaf. The reason is that if the node contains items with the same label, the entropy of the node will be zero and the entropy of all candidate splits will also be zero.
   Therefore, we can not gain any information by splitting the node, which meets the stopping criteria. Then we will stop at that node and it will be a leaf.

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.



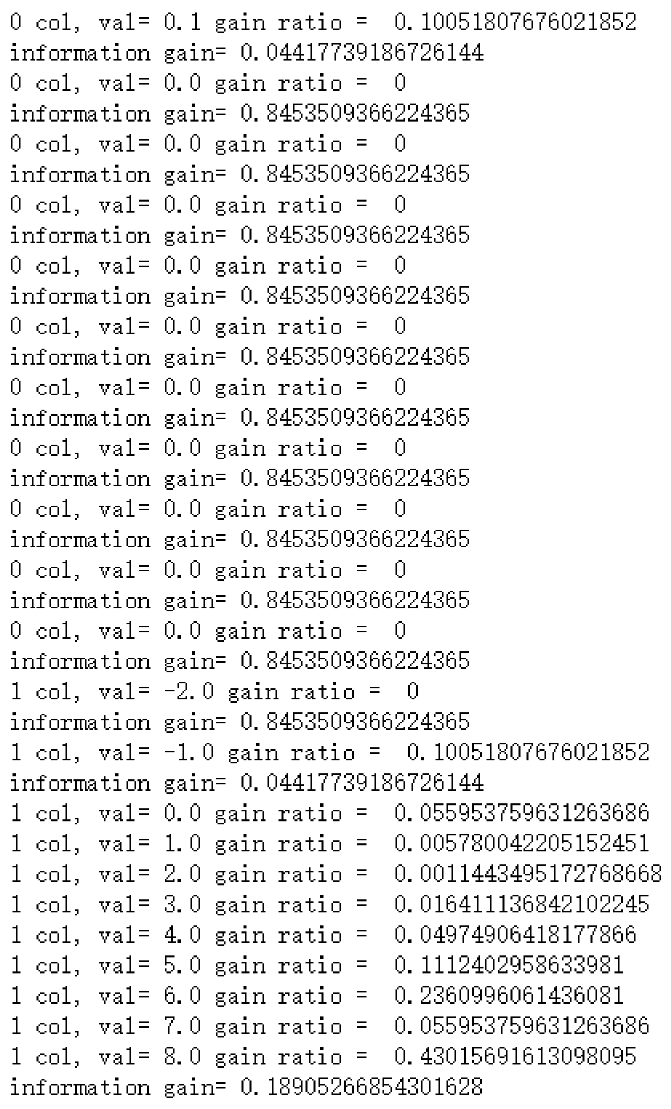
```
k2 = [[0, 0, 0],
      [0, 1, 1],
      [1, 0, 1],
      [1, 1, 0]]
```

Figure 1: training set

   Reason: According to the stop criteria, if all splits have zero gain ratio, the algorithm will stop. So in this training set, wherever the algorithm choose to split, it will always get zero gain ratio. But if we force a split, like $data[:, 0] \geq 1$, then the algorithm can easily continue splitting because we can set the condition like for $data[:, 0] \geq 1$, $data[:, 1] \geq 1 \rightarrow label = 0$ and $data[:, 1] < 1 \rightarrow label = 1$.

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.

   The candidate cut, corresponding gain ratio and mutual information gain are as follows:



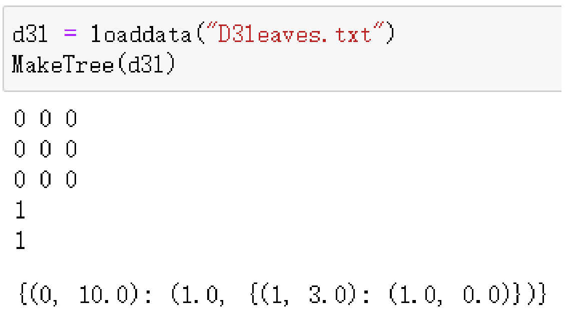
```
0 col, val= 0.1 gain ratio =  0.10051807676021852
information gain= 0.04417739186726144
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
0 col, val= 0.0 gain ratio =  0
information gain= 0.8453509366224365
1 col, val= -2.0 gain ratio =  0
information gain= 0.8453509366224365
1 col, val= -1.0 gain ratio =  0.10051807676021852
information gain= 0.04417739186726144
1 col, val= 0.0 gain ratio =  0.055953759631263686
1 col, val= 1.0 gain ratio =  0.005780042205152451
1 col, val= 2.0 gain ratio =  0.0011443495172768668
1 col, val= 3.0 gain ratio =  0.016411136842102245
1 col, val= 4.0 gain ratio =  0.04974906418177866
1 col, val= 5.0 gain ratio =  0.1112402958633981
1 col, val= 6.0 gain ratio =  0.2360996061436081
1 col, val= 7.0 gain ratio =  0.055953759631263686
1 col, val= 8.0 gain ratio =  0.43015691613098095
information gain= 0.18905266854301628
```

Figure 2: candidate cut/ gain ratio/ information gain

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.

   I use the dictionary in python to show the crude plaintext representation of the decision tree. The result is as follows:

[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.



```
d31 = loaddata("D31eaves.txt")
MakeTree(d31)

0 0 0
0 0 0
0 0 0
1
1

{(0, 10.0): (1.0, {(1, 3.0): (1.0, 0.0)})}
```

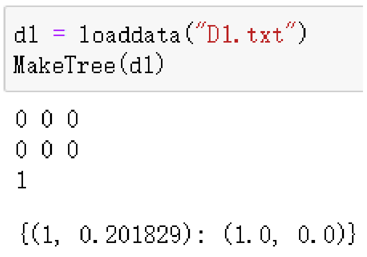
Figure 3: decision tree in dictionary

Rules: The tree is in $(col, value)$: $(left_subtree, right_subtree)$ or $(col, value)$: $(predict_1, predict_2)$ formula. The key of the dictionary is the feature i we choose and the value c. the first value(a.k.a the left subtree) in the tuple is a smaller dataset or the prediction of rows which satisfy the condition $x_i[col] \geq c$. And the second value(a.k.a the right subtree) in the tuple is a smaller dataset or the prediction of rows which cannot meet the condition.

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.

     The dictionary formula of the result is as follows:



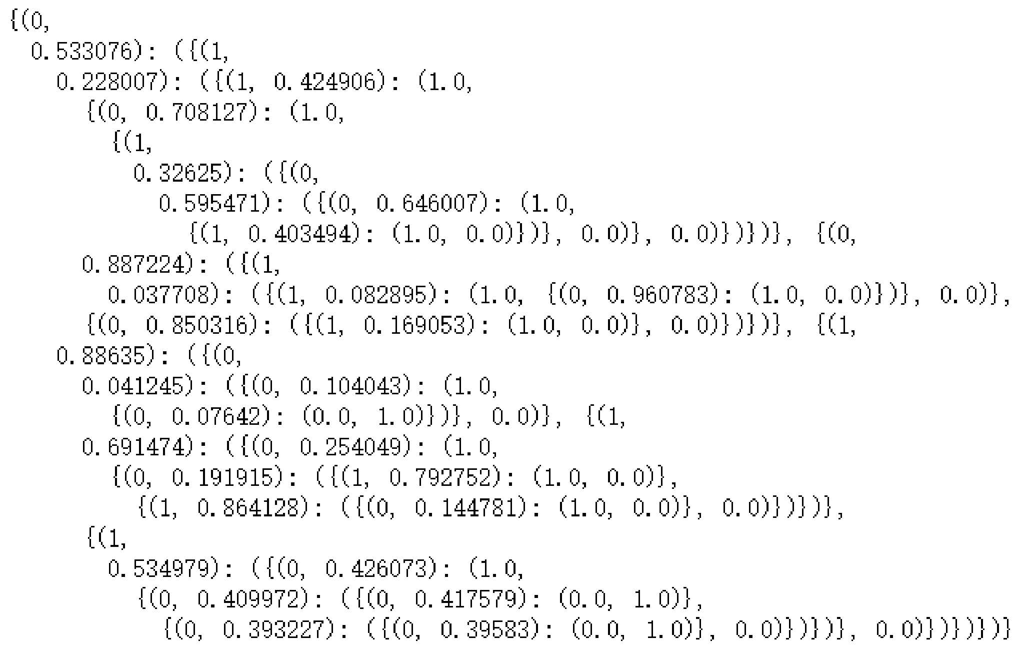
```
d1 = loaddata("D1.txt")
MakeTree(d1)

0 0 0
0 0 0
1

{(1, 0.201829): (1.0, 0.0)}
```

Figure 4: decision tree in dictionary

   - 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. Answer:To interpret, we only need to find all the rows i that satisfy Xi It's a simple decision tree. We only need to find all the rows i that satisfy $xi[1] \geq 0.201829$ and mark them as positive. For those rows which can not meet the condition above, we mark them as negative.
   - Build a decision tree on D2.txt. Show it to us.
     The dictionary formula of the result is as follows:



```
{(0,
  0.533076): ({(1,
    0.228007): ({(1, 0.424906): (1.0,
      {(0, 0.708127): (1.0,
        {(1,
          0.32625): ({(0,
            0.595471): ({(0, 0.646007): (1.0,
              {(1, 0.403494): (1.0, 0.0)})}, 0.0)}, 0.0)})})}, {(0,
      0.887224): ({(1,
        0.037708): ({(1, 0.082895): (1.0, {(0, 0.960783): (1.0, 0.0)})}, 0.0)},
      {(0, 0.850316): ({(1, 0.169053): (1.0, 0.0)}, 0.0)})})}, {(1,
    0.88635): ({(0,
      0.041245): ({(0, 0.104043): (1.0,
        {(0, 0.07642): (0.0, 1.0)})}, 0.0)}, {(1,
      0.691474): ({(0, 0.254049): (1.0,
        {(0, 0.191915): ({(1, 0.792752): (1.0, 0.0)},
          {(1, 0.864128): ({(0, 0.144781): (1.0, 0.0)}, 0.0)})})},
      {(1,
        0.534979): ({(0, 0.426073): (1.0,
          {(0, 0.409972): ({(0, 0.417579): (0.0, 1.0)},
            {(0, 0.393227): ({(0, 0.39583): (0.0, 1.0)}, 0.0)})})}, 0.0)})})})}
```

Figure 5: decision tree in dictionary

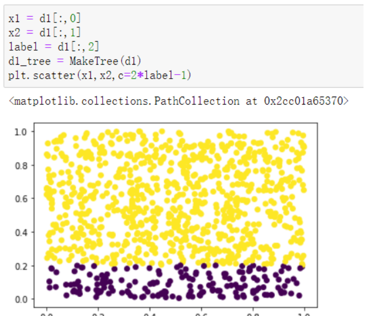
- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  Answer: This decision tree is very difficult to interpret with too many nodes. The only useful message is that there are about 60 nodes. That's why we need to visualize the decision tree.

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.

   The scatter plot of D1.txt is:

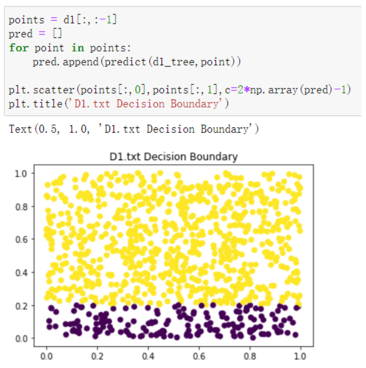


```
x1 = d1[:,0]
x2 = d1[:,1]
label = d1[:,2]
d1_tree = MakeTree(d1)
plt.scatter(x1,x2,c=2*label-1)
```

```
<matplotlib.collections.PathCollection at 0x2cc01a65370>
```

Figure 6: decision boundary

The decision boundary is:



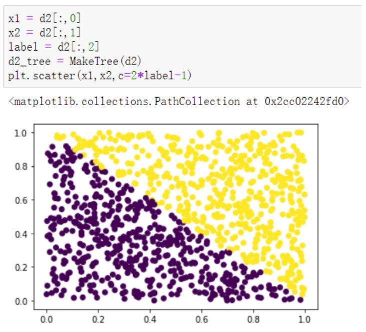
```
points = d1[:, :-1]
pred = []
for point in points:
    pred.append(predict(d1_tree, point))

plt.scatter(points[:, 0], points[:, 1], c=2*np.array(pred)-1)
plt.title('D1.txt Decision Boundary')
```

```
Text(0.5, 1.0, 'D1.txt Decision Boundary')
```

Figure 7: decision boundary

The scatter plot of D2.txt is:

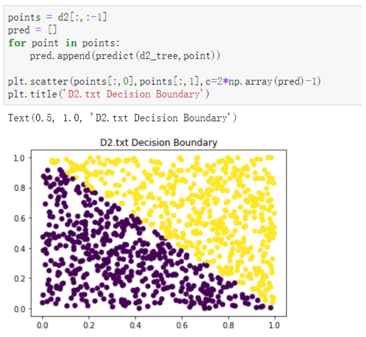


```
x1 = d2[:, 0]
x2 = d2[:, 1]
label = d2[:, 2]
d2_tree = MakeTree(d2)
plt.scatter(x1, x2, c=2*label-1)
```

```
<matplotlib.collections.PathCollection at 0x2cc02242fd0>
```

Figure 8: decision boundary

The decision boundary is:



```
points = d2[:, :-1]
pred = []
for point in points:
    pred.append(predict(d2_tree, point))

plt.scatter(points[:, 0], points[:, 1], c=2*np.array(pred)-1)
plt.title('D2.txt Decision Boundary')
```

```
Text(0.5, 1.0, 'D2.txt Decision Boundary')
```

Figure 9: decision boundary

For the decision tree, each node draw a line for certain feature, which means it is easy for it to find a horizontal or vertical boundary in the hypothesis space. In this case, the boundary of D1.txt is horizontal, and the boundary of D2.txt is from top left to bottom right. So the boundary of D2.txt is far more difficult for the decision tree to identity. That's why the size of your decision trees on D1 and D2 differ.

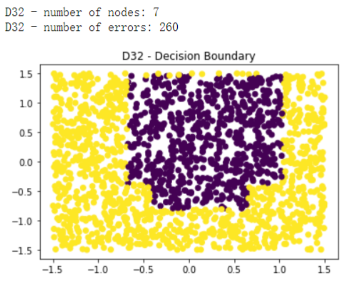
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

   - 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.
   - 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.

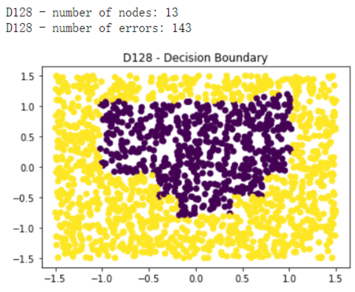
- For each D_n above, train a decision tree. Measure its test set error err_n. Show three things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n. This is known as a learning curve (a single plot). (3) Visualize your decision trees' decision boundary (five plots).

  For each tree trained from D32, D128, D512, D2042, D8192, the results are as follows:



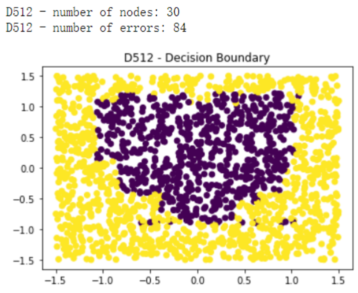
```
D32 - number of nodes: 7
D32 - number of errors: 260
```

Figure 10: node number/ error number/ decision boundary



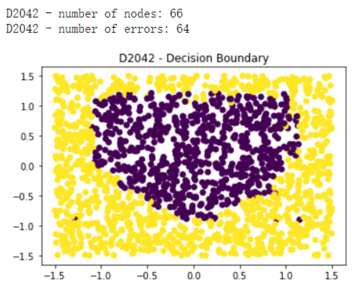
```
D128 - number of nodes: 13
D128 - number of errors: 143
```

Figure 11: node number/ error number/ decision boundary



```
D512 - number of nodes: 30
D512 - number of errors: 84
```

Figure 12: node number/ error number/ decision boundary



```
D2042 - number of nodes: 66
D2042 - number of errors: 64
```

Figure 13: node number/ error number/ decision boundary

```
D8192 - number of nodes: 127
D8192 - number of errors: 30
```

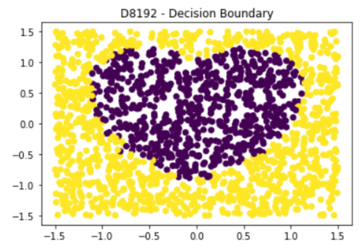


Figure 14: node number/ error number/ decision boundary

The relationship between node number and error number is as follows:

```
plt.plot(nodenum_list, error_list)
```

```
[<matplotlib.lines.Line2D at 0x2cc04e0aa60>]
```

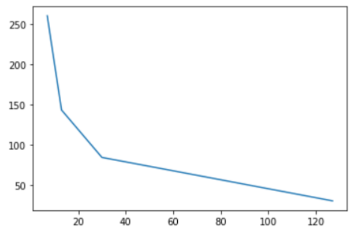


Figure 15: node number/ error number/ decision boundary

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.
Generally speaking, the larger the training set is, the more nodes in the decision tree and the more arccurate the model is. The list n, corresponding err_n and the plot n vs. err_n is as follows:

```
print('The order is "D8192,D2042,D512,D128,D32"')
print(n,err)
```

```
The order is "D8192,D2042,D512,D128,D32"
[233, 121, 59, 29, 17] [0.014380530973451378, 0.028761061946902644, 0.04922566371681414, 0.08573008849557517, 0.11227876106
```

```
plt.figure(figsize=(10, 10))
plt.plot(n,err)
plt.show()
```

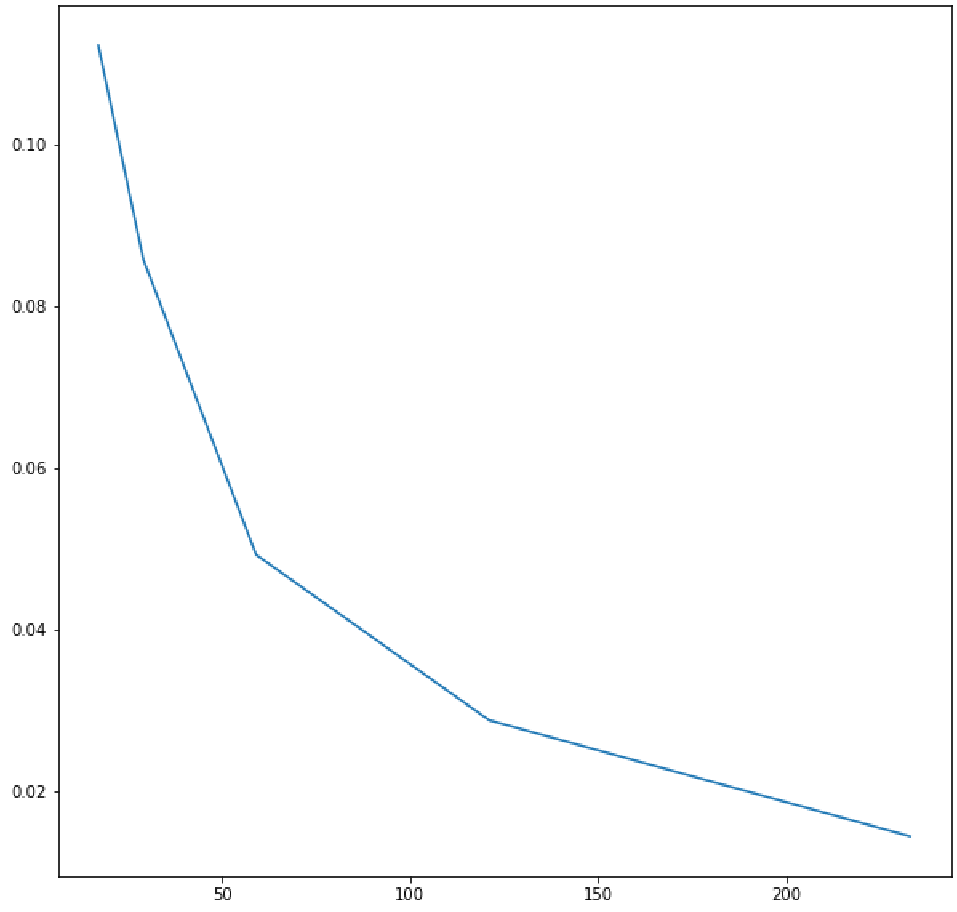


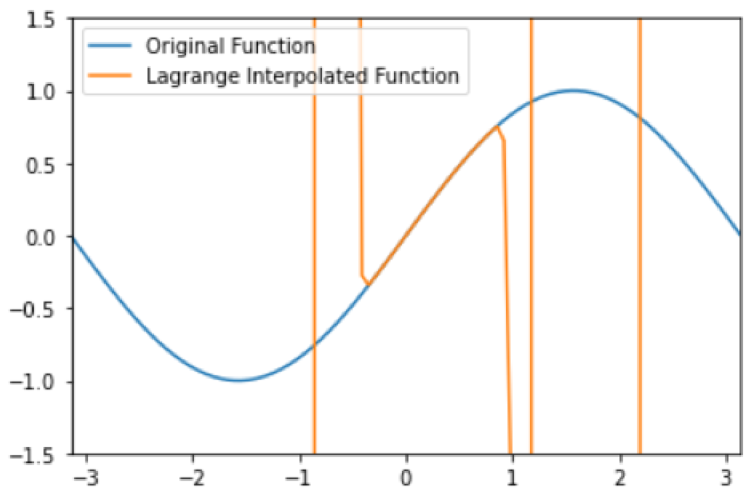
Figure 16: n, err_n and plot n vs. err_n

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. From the documents on https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.lagrange.html, we know that the function is numerically unstable. And the performance of the model on the training set is as follows:

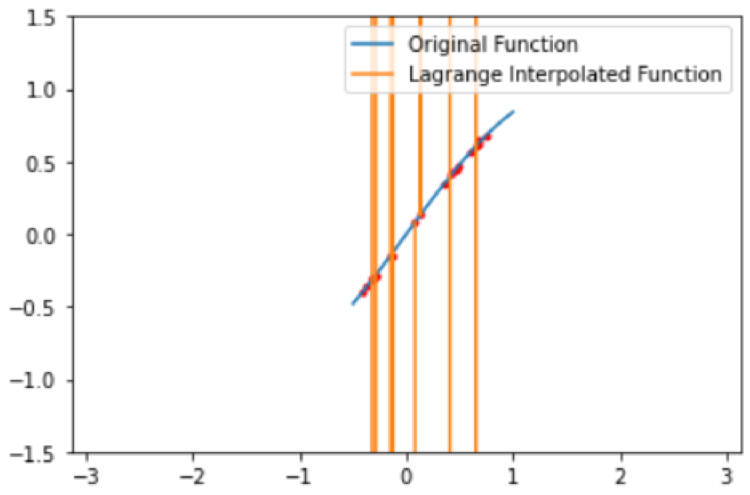


```
print(f'Train error  {std_dev}: {train_error:.6f}')
print(f'Test error  {std_dev}: {test_error:.6f}')
```

```
Train error  0.1: 17324947282373966333599853941367328886335332687887772957422204
449779357843456.000000
Test error  0.1: 122269016860194737569529118338426449544859208480467463981806581
9673447890944.000000
```

Figure 17: Lagrange interpolation

The model performs well in the interval of [-0.5,1]. But once it exceeds this range, the performance drops sharply. So we choose x from [-0.5,1] and add zero-mean Gaussian noisy. The result is as follows:



```
print(f'Train error  {std_dev}: {train_error:.6f}')
print(f'Test error  {std_dev}: {test_error:.6f}')
```

```
Train error  0.01: 68041095428698648063467364260969154285692123671596272605488401963194198250991
Test error  0.01: 3121478268732874160774155663381561470994850410777364218765115392.000000
```

Figure 18: Lagrange interpolation

The output of the model is very unstable. When I chose different standard deviation, I found that there is a positive correlation the stability of the model and the standard deviation