Assignment 1:
Model Comparison

Oindril Dutta
odutta3@gatech.edu

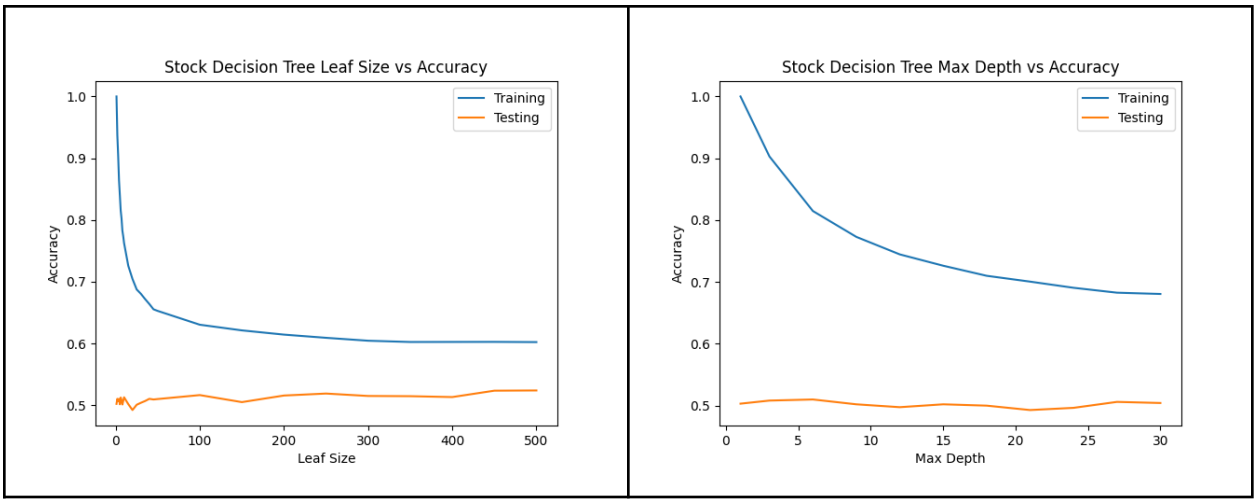
***Abstract*—For my assignment, I will be tackling two datasets I've worked with before in the past. The first is a dataset I created by hand: angles & magnitudes of the joints of a human body doing yoga poses, labeled with either Warrior II, Triangle, Tree, or None - crucial to the applicability of the model. The second is custom stock data, rearranged to be the close price of the stock for the last n days, normalized for those days, labeled with either buy or sell, based on if the n+1 close price is higher than the nth price.**

1 TIME-SERIES STOCK DATA SET

The first dataset is a twist on the standard close price of each stock over time. After working with stock data for a whole semester in CS 7646, I had an idea - instead of doing statistical models over the last n days to get an idea of whether the price will go up or down, why not train the model directly on the ebb and flow of the price for the last n days? The data is the price of the stock for the last n days, along with a label of "buy" or "sell" based on the price the next day, making it a classification problem. I believe this would be very interesting, and quite amazing if any model is able to get higher than 50% accuracy. It's not trivial at all, may perform completely differently on different models, and may not work! **In general, it doesn't work.**

1.1 Decision Tree

My Decision Tree implementation uses the GINI criterion using SKLearn.

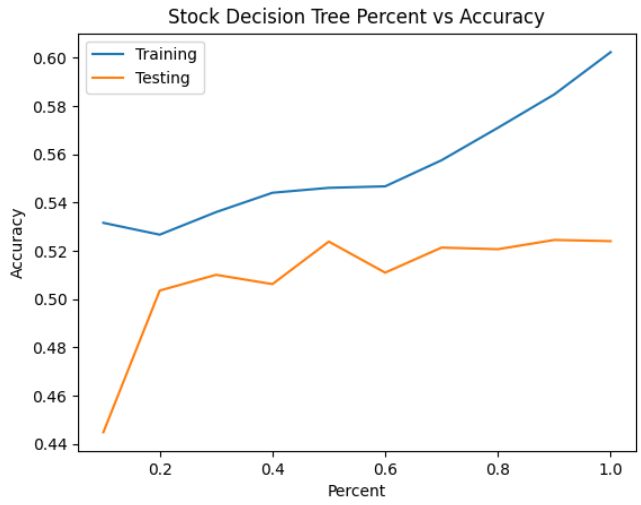


1.1.1 *Max Depth (Pruning)*

The Max Depth of the tree affects how deeply the tree can branch out to fit the dataset. In this case, the dataset isn't that great, so the testing accuracy is always terrible, but also, a lower max depth leads to overfitting, which is interesting because my understanding is that a higher max depth should actually allow the tree to overfit. In this case, I picked a final Max Depth of None since I couldn't understand what the best Max Depth should be from the graph.

1.1.2 *Min Leaves*

The leaf size is very indicative and matches my expectations. The minimum leaf size of 1 allows the decision tree to completely overfit the data, which makes sense because then it doesn't have to deal with grouping together multiple data points into the same leaf. As the leaf size increases, the testing accuracy gets marginally better, because the decision tree is less general, so I picked 500 for the final Min Leaves.

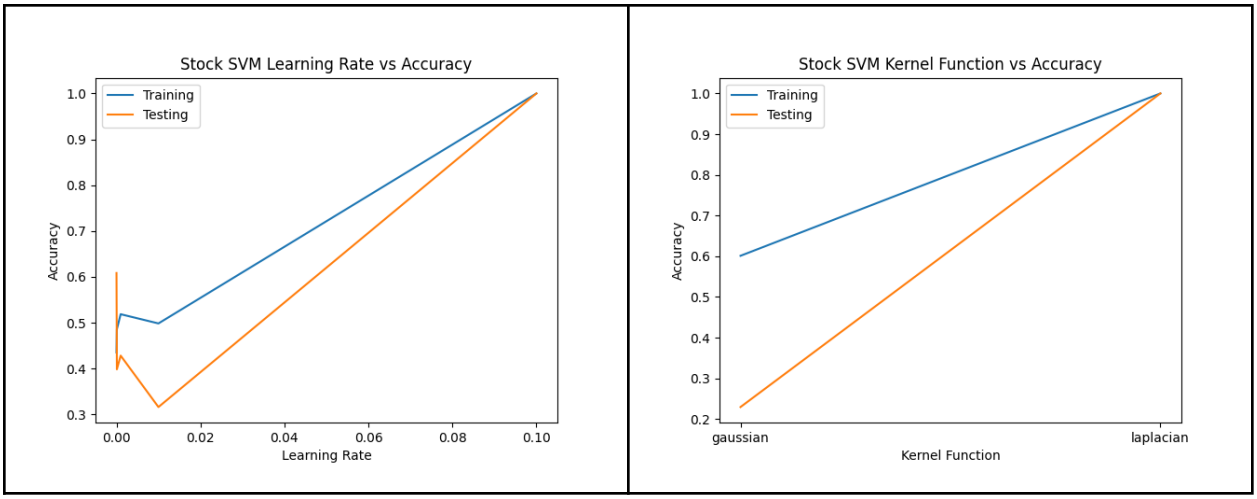


Using a Max Depth of None and a Min Leaves of 500, the decision tree performed better with more data, reaching the highest accuracy of 52% on the testing data, as good as random. Either Decision Trees aren't the best for this data/problem, or this data itself isn't great.

~~1.2 Decision Tree with Boosting - N/A~~

1.3 Support Vector Machine

I went over what support vector machines are conceptually, but I still don't understand them completely, or why I get the results I do. I implemented it with Keras, using a RandomFourierFeatures layer in the model to simulate a Support Vector Machine.



1.3.1 *Kernel Function*

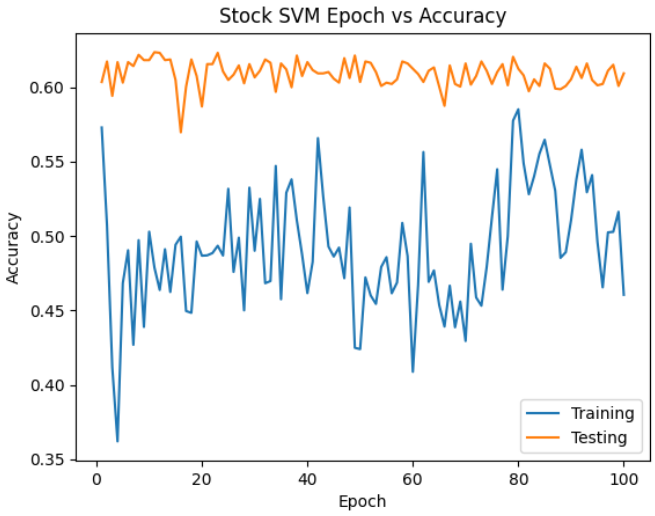
I tried two Kernel Functions, the Gaussian and the Laplacian. I don't understand the differences between them, but for the stock dataset the Laplacian kernel function did much better according to the graph, so I used that for the final graph.

```
Gaussian: K(x, y) == exp(- square(x - y) / (2 * square(scale)))

Laplacian: K(x, y) = exp(-abs(x - y) / scale))
```

1.3.2 *Learning Rate*

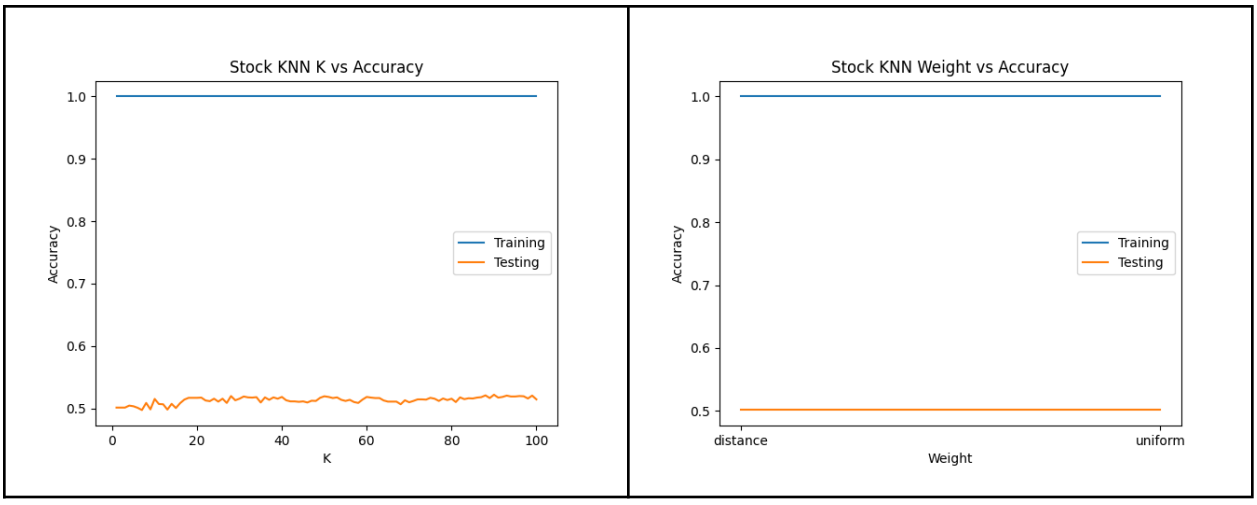
I used different learning rates to see if it would handle the data better, and surprisingly, the fastest learning rate did the best with 100 epochs in the testing and the training, which I also didn't understand why, but I used a 0.1 learning rate for the final.



When it came down to replicating the results, however, the model did poorly (suggesting maybe the previous two graphs were flukes), even though it scored the highest of all the other models for testing accuracy on the stock data, which means it might have actually found the data points best needed to generalize itself with the biggest margin to get a slightly better than random chance accuracy.

1.4 K Nearest Neighbors

I used the KNN model from SKLearn to implement this.



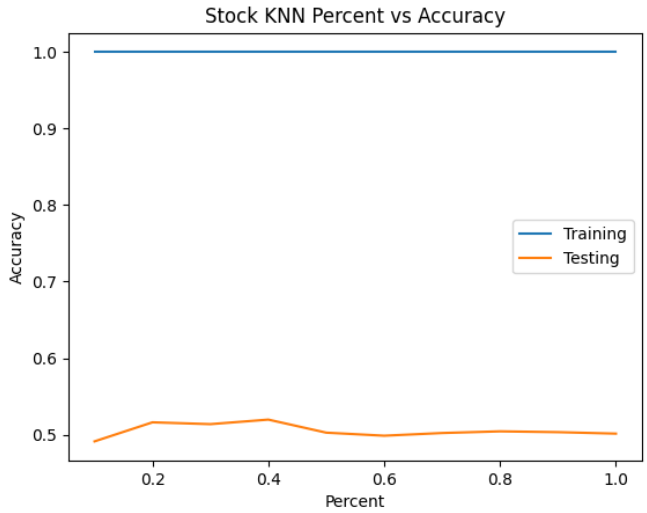
1.4.1 *K*

This isn't a great representation of the KNN model, since it only barely does better on the test dataset with a higher K, but typically, a higher K should lead to better generalizability of the

model. Not surprisingly, it's overfitting for the training data, and weirdly enough it doing that regardless of the K, I'd think a higher K in the training should contribute to less overfitting.

1.4.2 *Weight (Distance based Weighing)*

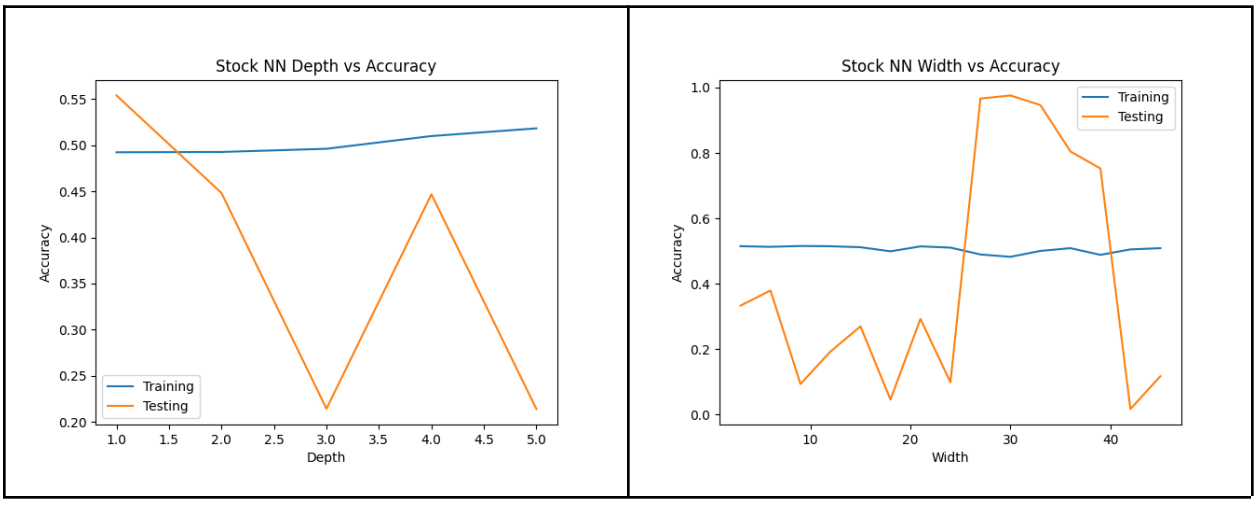
There was a choice of using either uniform weighing between points or distance-based weighing, in this case, it didn't seem to make a difference, but in general distance-based weighing, given a proper distance function, should help.



KNN was not a good fit for the stock data, not doing well regardless of the percentage of data passed in.

1.5 Neural Net

I used a Keras Neural Net with a variable width and depth dense layers.

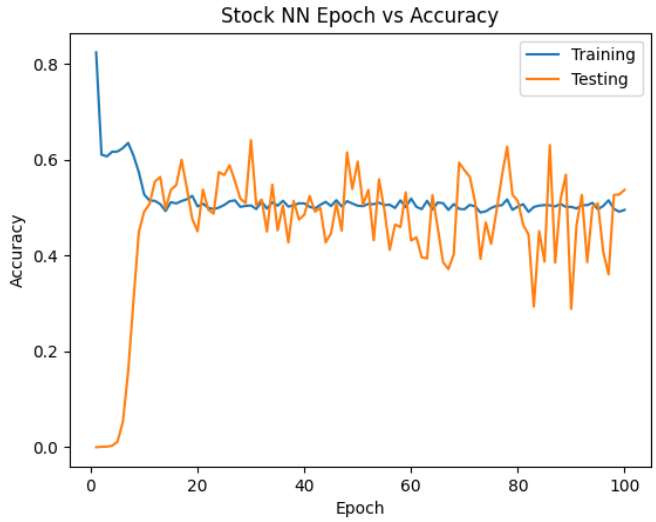


1.5.1 *Depth*

The depth is the number of layers between the input and output layer. The best was a depth of 5 across multiple runs.

1.5.2 *Width*

The width is the number of neurons in each layer between the input and output. The best was a width of 30, exactly stock days * 2, across multiple runs.



The NN model didn't fare well on the Stock Dataset either, however, as the epochs went on it got more and more unstable, suggesting it may have done better if there were more epochs it was allowed to train over.

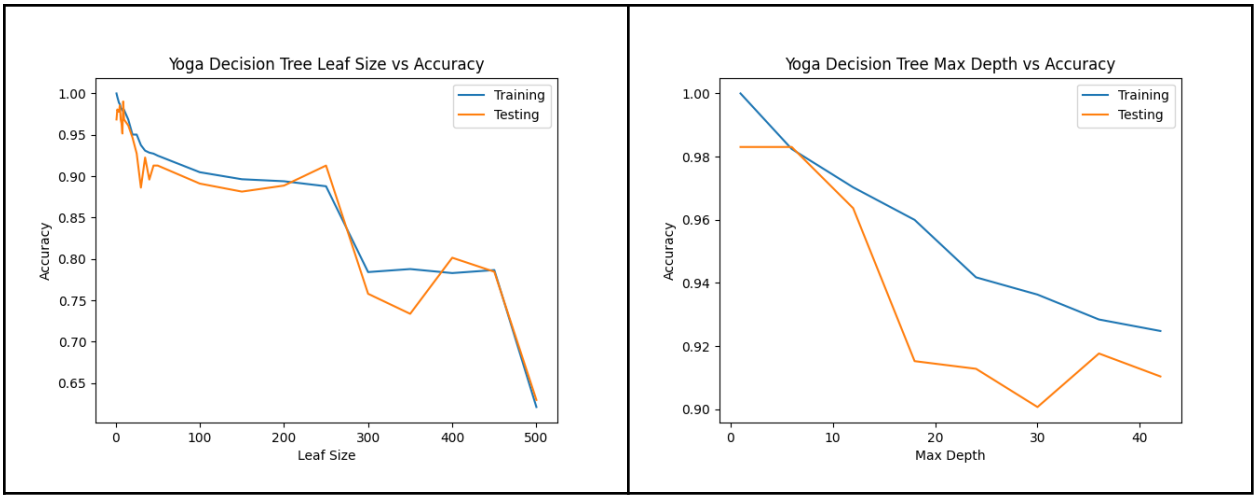
2 YOGA POSES DATA SET

The second dataset is also quite quirky, especially because it's hand generated. It's around 2000 rows of numerical skeletal data, labeled with a yoga pose. I got this data by actually inputting in youtube URLs, along with timestamps, and each frame within those timestamps was turned into a frame, cropped and scaled down to 100x100 images, ran them through OpenPose, that outputted a point map skeleton of the joints, which I then converted to magnitudes and angles, which I will use as the data. To see how I got the data, look at the JSON file in the yoga folder after cloning my repo, look for the URL under the objects in the history key, to see the videos I captured frames from. To see the frames themselves, you can decode the base64 string images in the objects in the frame key. I know for a fact this should perform well with a neural net, but it would be interesting to see if that translates to other models. It's definitely non-trivial because I

can't just hand-code an algorithm to solve it, I've tried before, and it may be pretty interesting when it comes down to how it does on different algorithms.

2.1 Decision Tree

when it comes down to how

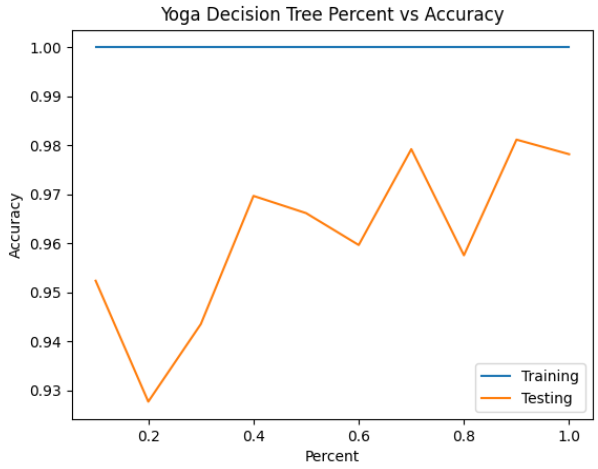


2.1.1 *Max Depth Pruning*

Body

2.1.2 *Leaf Size*

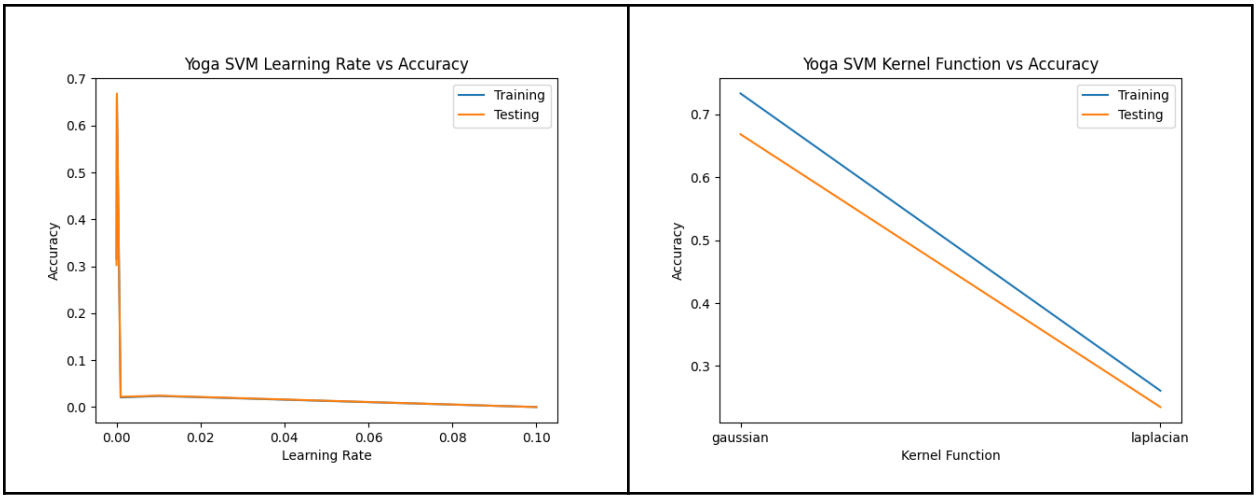
Body



Body

~~2.2 Decision Tree with Boosting - N/A~~

2.3 Support Vector Machine

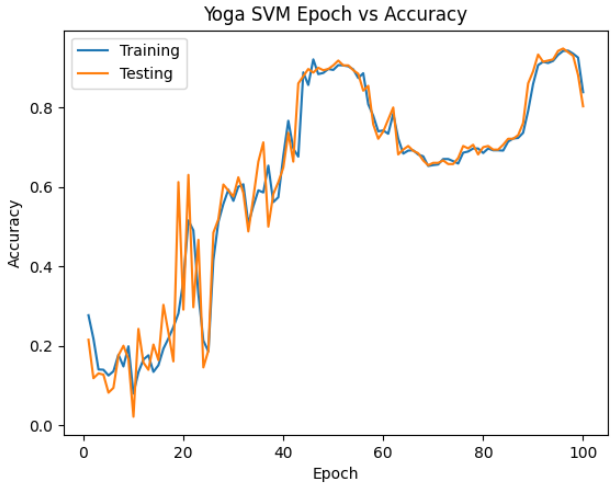


2.3.1 *Loss Function*

Body

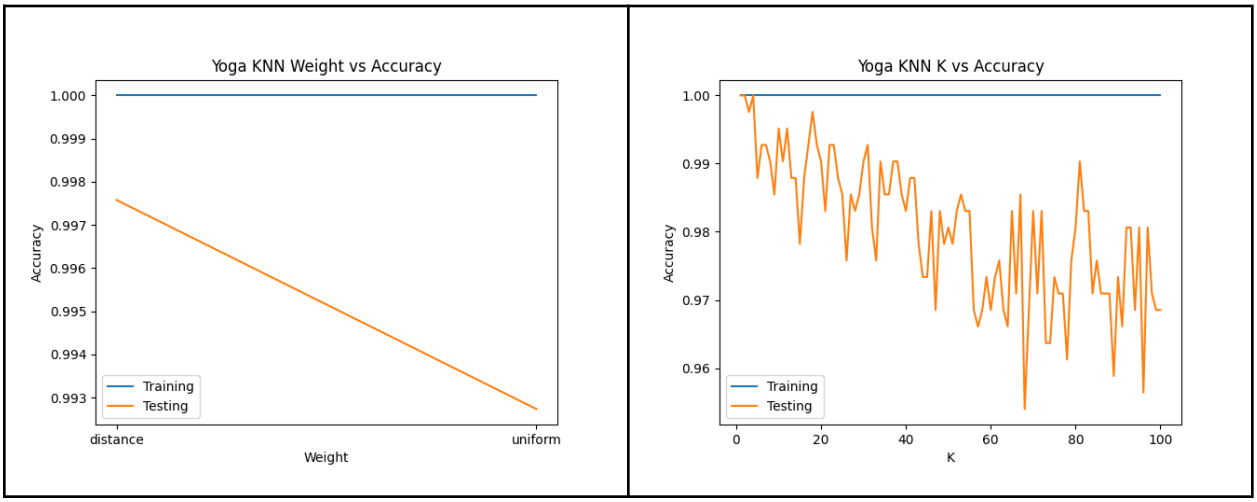
2.3.2 *Learning Rate*

Body



Body

2.4 K Nearest Neighbors

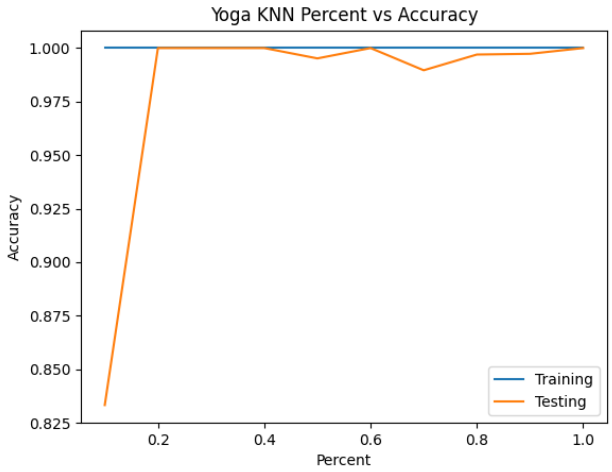


2.4.1 *K*

Body

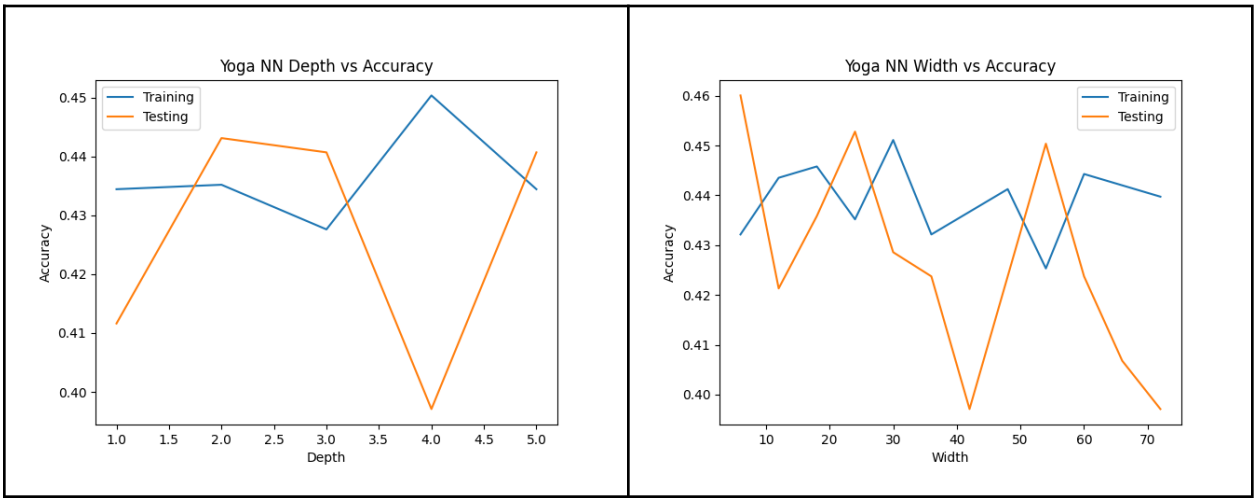
2.4.2 *Weight*

Body



Body

2.5 Neural Net

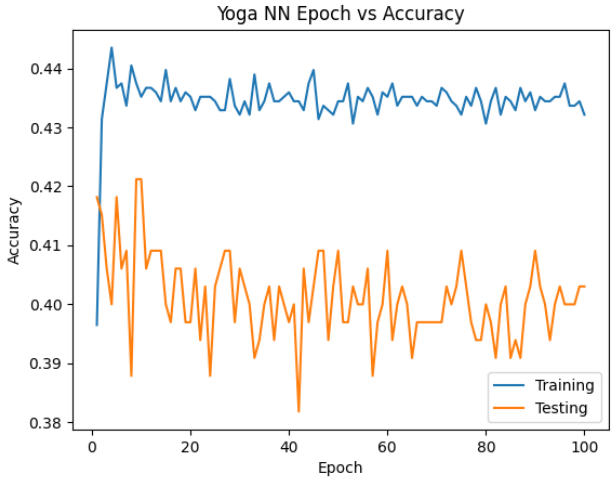


2.5.1 *Depth*

Body

2.5.2 *Width*

Body



Body

Figure 1— add a caption