# Privacy Preserving Decision Tree Prediction

Reem Younis, Assia Khateeb, Atheer Abo Foul, Aya Miari
Lecturer : Dr. Adi Akavia
Laboratory in Privacy Preserving Machine Learning, University of Haifa
Email: reembyounis@gmail.com, assia.khteb@gmail.com, 19aether6@gmail.com, aia-m-211@hotmail.com

July 7, 2021

## Contents

**Abstract.**
In part b we implement cleartext soft decision tree prediction algorithm; algorithm specified in Akavia et al ECML'2020.

# 1 Introduction

Reference to our Github repo.
Part b of the lab is done by dividing the problem into multiple phases, each one we called it pre-processing phase.

- First pre-procssing is scaling the data.
- Second pre-procssing is preparing the polynom which is required for Algorithm 1.
- Third pre-procssing is building a tree with 1-hot encoding labels.

We predict on set of samples using Algorithm 1. Then, we present the main results of Algorithm 1 and compare it to scikit learn results.

# 2 Preprocessing Phase

In this section, by using preprocessing, input data is processed to produce an output data that is used in our program.

## 2.1 SCALING

We used the sklearn.preprocessing package which provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

We start by re-scaling each value in the given data set, to a value in range $[-1, 1]$ (according to the article) using MinMaxScaler package from sklearn. The general formula to rescale a range between a of values [a, b] is given as:

$$x' = a + \frac{(x - min(x))(b - a)}{max(x) - min(x)} \tag{1}$$

In our case $[a, b]$ is $[-1, 1]$ as explained above.
Reference to Feature Scaling in Wikipedia: Feature Scaling

Figure 1: DATA BEFORE SCALING

```
#Load Data and store it into pandas DataFrame objects
iris = load_iris()
X = pd.DataFrame(iris.data[:, :], columns=iris.feature_names[:])
print(X)
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

Figure 2: DATA AFTER SCALING

```
# preprocessing 1 - Feature scaling
# rescale a range between an arbitrary set of values [a, b] where a=-1, b=1
scaler = MinMaxScaler(feature_range=(-1, 1)) # build the scaler model
X_rescaled_features = scaler.fit_transform(X)
X_rescaled_features = pd.DataFrame(X_rescaled_features[:, :], columns=iris.feature_names[:])
print(X_rescaled_features)
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0            -0.555556          0.250000          -0.864407         -0.916667
1            -0.666667         -0.166667          -0.864407         -0.916667
2            -0.777778          0.000000          -0.898305         -0.916667
3            -0.833333         -0.083333          -0.830508         -0.916667
4            -0.611111          0.333333          -0.864407         -0.916667
..                 ...               ...                ...               ...
145           0.333333         -0.166667           0.423729          0.833333
146           0.111111         -0.583333           0.355932          0.500000
147           0.222222         -0.166667           0.423729          0.583333
148           0.055556          0.166667           0.491525          0.833333
149          -0.111111         -0.166667           0.389831          0.416667

[150 rows x 4 columns]
```

We next use train_test_split model from sklearn to Split data set into random train and test subsets.
Using the train parameter we build the decision tree using DecisionTreeClassifier package from sklearn, choosing 4 as the tree's max depth.

### 2.1.1 Code

```
def scaling(X):
    scaler = MinMaxScaler(feature_range=(-1, 1))
    X_rescaled_features = scaler.fit_transform(X)
    return X_rescaled_features
```

## 2.2 POLYNOM

We construct a low-degree polynomial approximation in order to approximate the step function, aiming to replace it with a soft step function.
This is done by using mean square integral solution which is the soft step function :

$$\phi = \min_{p \in p_n} \int_{-2}^{2} (I_0(x) - p(x))^2 \, dx \tag{2}$$

Then, by adding an importance to weight of the approximation interval we maintain the sensitivity to error in the approximation is uniform over the domain. This leads to the optimization problem given in the article :

$$\phi = \min_{p \in p_n} \int_{-2}^{2} (I_0(x) - p(x))^2 w(x) \, dx \tag{3}$$

### 2.2.1 Code

We present our code for the polynomial approximation:

```
def polynom(degree, window):
    X = np.linspace(-2, 2, num=201)
    y1 = np.zeros(100)
    y2 = np.ones(101)
    Y = np.concatenate((y1, y2), axis=None)
    # weights functions
    w1 = np.concatenate((window * (np.ones(75)), np.zeros(51)), axis=None)
```

```
w2 = window * np.ones(75)
weight = np.concatenate((w1, w2), axis=None)
pf = np.polyfit(X, Y, degree, w=weight)
phi = np.poly1d(pf)
return phi
```

Using sklearn numpy model we used linspace function to create an evenly spaced samples which are calculated over an interval[start,stop], the returned value is stored in parameter X.
Followed by parameter Y, the step function which maps the values $[1, 100]$ to 0 and values $[101, 201]$ to 1.

X represents a numpy array with values in range $[-2, 2]$ with 201 samples.
If we choose window to be equal to 0.25 then, the array X will be divided to 3 ranges: $[-2 : -0.25]$, $[-0.25 : 0.25]$ and $[0.25 : 2]$. The size of ranges $[-2 : -0.25]$ + $[0.25 : 2]$ is 7/2. The values in ranges $[1 : 75]$ and $[127 : 201]$ have a weight of 2/7 according to the equation:

$$\int_{-2}^{2} w(x)\, dx \; = 1 \tag{4}$$

while the values in range $[76 : 126]$ have weights of 0. Using polyfit model, X is fitted to Y with degree = 34 and the calculated weights above.

We perform polynomial fitting on data set using polyfit model which returns a vector of coefficients p that minimizes the squared error. Then we create a polynomial model using poly1d function, which it's return value indicates whether the polynomial's coefficients powers are in a decreasing order. The result is stored in parameter phi.

## 2.3 Build Tree With 1-Hot Encoding Labels

We start building our tree by calling the builtTree function in file my_tree.py.
Each tree's inner node has a field for threshold, feature and node_id.
Parameters in builtTree function :
lenHot = length of label's value.
index_of_max = index of argmax value.
Using these two parameters, leaf array is built, so it's size is equal to the size of lenHot array. It's initialized with zeros, and ones in index_of_max.
Thus, we get an array of values as a form of 1 hot encoding.
Example:

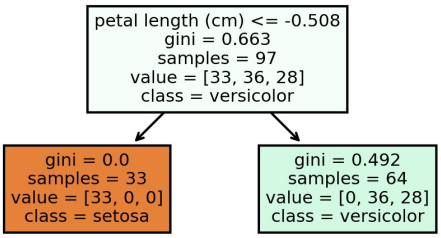


Figure 3:
Subtree labels before 1-hot encoding

Labels of the subtree after 1-hot encoding.
leaf = [1. 0. 0.]
leaf = [0. 1. 0.]

builtTree function This function receives as input, the tree and node_id which is initialized to 0. Tree's nodes are numbered by post-order traversal.
Recursively, leaves values are updated to 1 hot encoding format.

printTree function:This function receives as input, the tree and tree's depth which is initialized to 0.
For every inner node, threshold and feature are printed. However, for every leaf, only 1 hot value is printed.

printTree function recognizes whether the node is an inner node or leaf using $isinstance(leaf, np.ndarray)$ which returns if there's an array in the current node, if True then it's a leaf.

# 3 Algorithm 1

### 3.0.1 Code

```
def Tree_Predict(T, x, phi):
    if T is None:
        return
    feature, threshold, leaf, left, right = T.getNode()
    if isinstance(leaf, np.ndarray):
        return leaf
    else:
        return (phi(x[feature] - threshold)) * Tree_Predict(right, x, phi) +
        + (phi(threshold - x[feature])) * Tree_Predict(left, x, phi)
```

Akavia's algorithm traverses all paths in the tree and computes a weighted combination of all the leaves values, where each leaf value is the 1-hot encoding of the label associated with the leaf. The output is a length L vector assigning a likelihood score to each label, which is in turn interpreted as outputting the label with the highest score.

# 4 Accuracy

The accuracy is calculated as the percentage of correct classification on test samples.

```
algorithm1_score = (counter / len(res_vec))*100
```

Counter is equal to the sum of correct classifications and res_vec is equal to the sum of all classifications.
To know if current classification is correct or not we compare it to Y target value of the relevant sample if they are eqaul then the prediction of algorithm 1 is correct .

test samples are decided to sent to algorithm 1 as 35% of the samples in the data-set and they are chosen randomly from the data-set.
test samples are assigned to variable X_test.
So algorithm 1 predict over all test samples, after that we calculate accuracy according to the percentage of correct predictions on test samples that algorithm 1 predict.

```
for x in X_test:
    res = algorithm1_predict(myTree, x, phi)
    res_vec.append(res)
```

We trained trees up to depth 6 and compare algorithm 1 prediction's accuracy vs. scikit learn prediction's accuracy for three data sets: iris, wine and cancer.

Table 1: iris

| **Tree depth** | Algorithm 1 acc | scikit learn acc |
|---|---|---|
| **0** | 22.64151 | 22.64151 |
| **1** | 62.26415 | 62.26415 |
| **2** | 94.33962 | 94.33962 |
| **3** | 86.7925 | 96.22642 |
| **4** | 100 | 96.22642 |
| **5** | 94.3396 | 96.22642 |
| **6** | 98.1132 | 94.33962 |

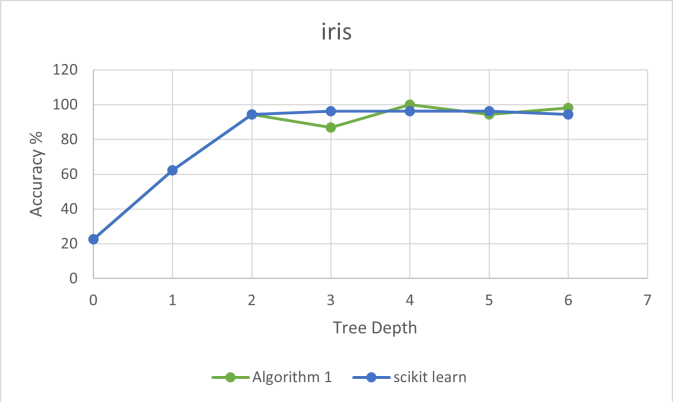


Figure 4:
Accuracy of ours vs. Scikit-learn on iris dataset and tree depth 0-6

Table 2: wine

| **Tree depth** | Algorithm 1 acc | scikit learn acc |
|---|---|---|
| **0** | 38.09524 | 38.09524 |
| **1** | 50.79365 | 50.79365 |
| **2** | 90.4762 | 87.30159 |
| **3** | 92.0635 | 90.47619 |
| **4** | 90.4762 | 88.88889 |
| **5** | 93.6508 | 92.06349 |
| **6** | 96.8254 | 92.06349 |

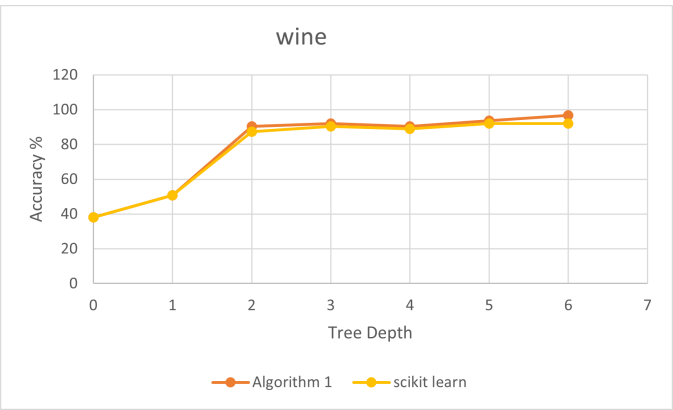


Figure 5:
Accuracy of ours vs. Scikit-learn on Wine dataset and tree depth 0-6

Table 3: cnacer

| **Tree depth** | Algorithm 1 acc | scikit learn acc |
|---|---|---|
| **0** | 61.5 | 61.5 |
| **1** | 87.5 | 87.5 |
| **2** | 94.5 | 91 |
| **3** | 96 | 96 |
| **4** | 94.5 | 91.5 |
| **5** | 92 | 92 |
| **6** | 96.5 | 92 |

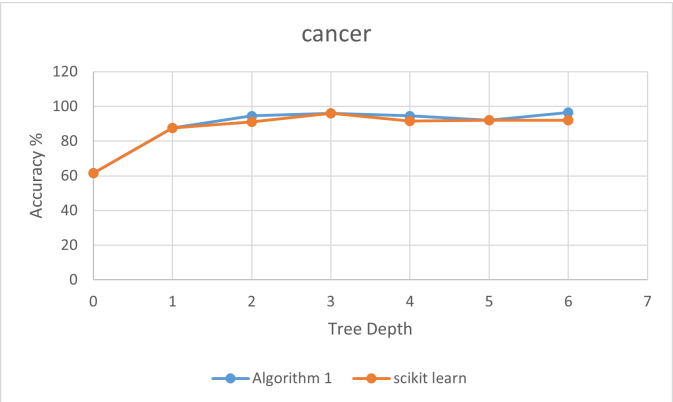


Figure 6:
Accuracy of ours vs. Scikit-learn on Cancer dataset and tree depth 0-6

# 5 Results

Example of results we got :

data_type = iris
max depth = 4
scikit_learn_score = 92.45283 %

WINDOW = 0

window = 0
polynom order = 0
run time = 0.0180 seconds
algorithm1_score = 32.0755 %

---

window = 0
polynom order = 1
run time =0.0150 seconds
algorithm1_score = 90.5660 %

---

window = 0
polynom order = 2
run time =0.0180 seconds
algorithm1_score = 90.5660 %

---

window = 0
polynom order = 3
run time = 0.0190 seconds
algorithm1_score = 92.4528 %

---

.
.
.
window = 0
polynom order = 33
run time = 0.0349 seconds
algorithm1_score = 92.4528 %

---

WINDOW = 0.25

window = 0.25
polynom order = 0
run time = 0.0439 seconds
algorithm1_score = 32.0755 %

.
.
.
window = 0.25
polynom order = 33
run time = 0.0499 seconds
algorithm1_score = 92.4528 %

---

WINDOW = 0.5

window = 0.5
polynom order = 0
run time = 0.0429 seconds
algorithm1_score = 32.0755 %

.
.
.
window = 0.5
polynom order = 33
run time = 0.0499 seconds
algorithm1_score = 92.4528 %

---

WINDOW = 0.75

window = 0.75
polynom order = 0
run time = 0.0259 seconds
algorithm1_score = 32.0755 %

.
.
.
window = 0.75
polynom order = 33
run time = 0.0289 seconds
algorithm1_score = 92.4528 %

**For each window [0, 0.25, 0.5, 0.75] We calculate the algorithm's 1 accuracy for every polynom's degree in range [0,33]. Then we arranged all the results in tables ( table 4 to 15).**

## 5.1 Results For Cancer Data-Set

Cancer Tree:

feature = 22 threshold = -0.41192
    feature = 27 threshold = 0.28384
        feature = 27 threshold = -0.23814
            feature = 28 threshold = -0.99980
                leaf = [1. 0.]
                leaf = [0. 1.]
            feature = 1 threshold = -0.23266
                leaf = [0. 1.]
                leaf = [1. 0.]
        leaf = [1. 0.]
    feature = 6 threshold = -0.65742
        feature = 1 threshold = -0.38282
            leaf = [0. 1.]
            feature = 15 threshold = -0.72109
                leaf = [1. 0.]
                leaf = [0. 1.]
        feature = 7 threshold = -0.57465
            feature = 9 threshold = -0.49326
                leaf = [1. 0.]
                leaf = [0. 1.]
            feature = 1 threshold = -0.70172
                leaf = [1. 0.]
                leaf = [1. 0.]

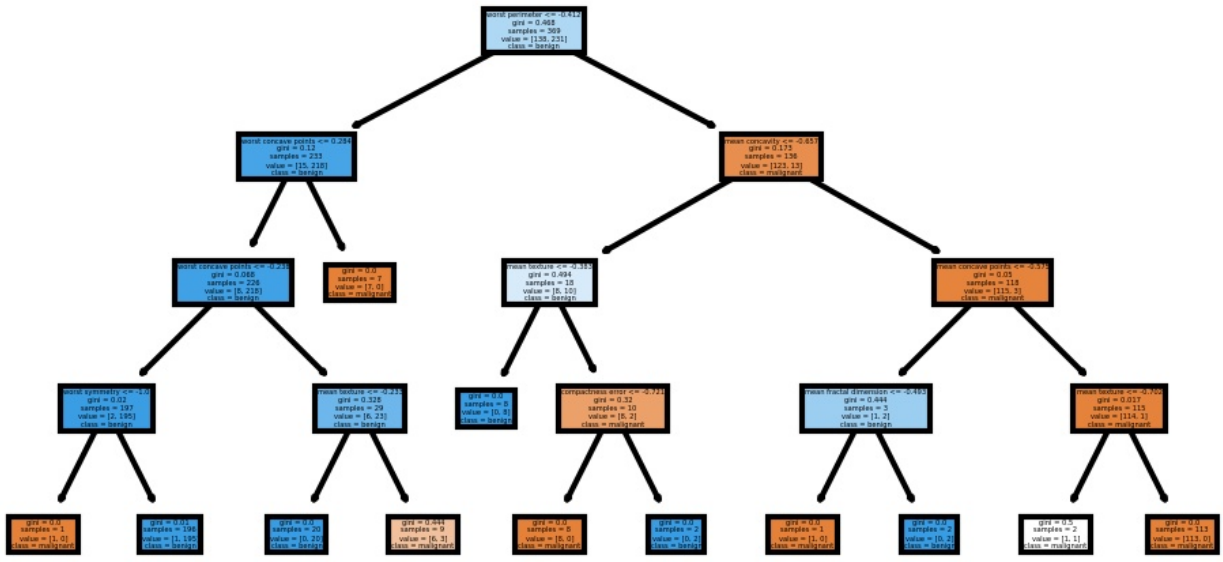

Figure 7: Cancer Tree Figure (Before 1 Hot Encoding) With Max Depth = 4

Table 4:

| **Dataset: Cancer** | | | |
|---|---|---|---|
| **scikit learn score = 94.50000 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0 | 0 | 0.0788 | 37 |
| | 1 | 0.0798 | 75.5 |
| | 2 | 0.0927 | |
| | 3 | 0.1087 | 92 |
| | 4 | 0.1077 | |
| | 5 | 0.1197 | 92.5 |
| | 6 | 0.1137 | |
| | 7 | 0.1117 | 95.5 |
| | 8 | 0.1137 | |
| | 9 | 0.1177 | |
| | 10 | 0.1027 | |
| | 11 | 0.1137 | 96 |
| | 12 | 0.1117 | |
| | 13 | 0.1167 | 96.5 |
| | 14 | 0.1117 | |
| | 15 | 0.1227 | 97 |
| | 16 | 0.1206 | |
| | 17 | 0.1316 | 96.5 |
| | 18 | 0.1326 | |
| | 19 | 0.1247 | |
| | 20 | 0.1426 | |
| | 21 | 0.1416 | 96 |
| | 22 | 0.1326 | |
| | 23 | 0.1277 | 96.5 |
| | 24 | 0.1526 | |
| | 25 | 0.1307 | 96 |
| | 26 | 0.1735 | |
| | 27 | 0.1706 | |
| | 28 | 0.167 | |
| | 29 | 0.128 | |
| | 30 | 0.148 | |
| | 31 | 0.152 | |
| | 32 | 0.156 | |
| | 33 | 0.144 | |

Table 5:

| **Dataset: Cancer** | | | |
|---|---|---|---|
| **scikit learn score =94.50000 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.25 | 0 | 0.084 | 37 |
| | 1 | 0.08 | 75.5 |
| | 2 | 0.104 | |
| | 3 | 0.1 | 92 |
| | 4 | 0.092 | |
| | 5 | 0.104 | 92.5 |
| | 6 | 0.104 | |
| | 7 | 0.092 | 95.5 |
| | 8 | 0.116 | |
| | 9 | 0.096 | |
| | 10 | 0.1 | |
| | 11 | 0.12 | 96 |
| | 12 | 0.104 | |
| | 13 | 0.12 | 96.5 |
| | 14 | 0.128 | |
| | 15 | 0.108 | 97 |
| | 16 | 0.12 | |
| | 17 | 0.12 | 96.5 |
| | 18 | 0.144 | |
| | 19 | 0.128 | |
| | 20 | 0.128 | |
| | 21 | 0.124 | 96 |
| | 22 | 0.136 | |
| | 23 | 0.144 | 96.5 |
| | 24 | 0.124 | |
| | 25 | 0.164 | 96 |
| | 26 | 0.136 | |
| | 27 | 0.14 | |
| | 28 | 0.136 | |
| | 29 | 0.144 | |
| | 30 | 0.132 | |
| | 31 | 0.152 | |
| | 32 | 0.168 | |
| | 33 | 0.16 | |

Table 6:

| Dataset: Cancer | | | |
|---|---|---|---|
| **scikit learn score = 94.50000 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.5 | 0 | 0.08 | 37 |
| | 1 | 0.104 | 75.5 |
| | 2 | 0.08 | |
| | 3 | 0.104 | 92 |
| | 4 | 0.092 | |
| | 5 | 0.084 | 92.5 |
| | 6 | 0.112 | |
| | 7 | 0.1 | 95.5 |
| | 8 | 0.096 | |
| | 9 | 0.12 | |
| | 10 | 0.096 | |
| | 11 | 0.104 | 96 |
| | 12 | 0.124 | |
| | 13 | 0.1 | 96.5 |
| | 14 | 0.1 | |
| | 15 | 0.2 | 97 |
| | 16 | 0.128 | |
| | 17 | 0.132 | 96.5 |
| | 18 | 0.116 | |
| | 19 | 0.136 | |
| | 20 | 0.124 | |
| | 21 | 0.12 | 96 |
| | 22 | 0.136 | |
| | 23 | 0.14 | 96.5 |
| | 24 | 0.14 | |
| | 25 | 0.136 | 96 |
| | 26 | 0.132 | |
| | 27 | 0.136 | |
| | 28 | 0.14 | |
| | 29 | 0.156 | |
| | 30 | 0.148 | |
| | 31 | 0.16 | |
| | 32 | 0.148 | |
| | 33 | 0.152 | |

Table 7:

| **Dataset: Cancer** | | | |
|---|---|---|---|
| **scikit learn score = 94.50000 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.75 | 0 | 0.084 | 37 |
| | 1 | 0.1 | 75.5 |
| | 2 | 0.08 | |
| | 3 | 0.108 | 92 |
| | 4 | 0.088 | |
| | 5 | 0.084 | 92.5 |
| | 6 | 0.104 | |
| | 7 | 0.104 | 95.5 |
| | 8 | 0.092 | |
| | 9 | 0.116 | |
| | 10 | 0.104 | |
| | 11 | 0.112 | 96 |
| | 12 | 0.104 | |
| | 13 | 0.144 | 96.5 |
| | 14 | 0.12 | |
| | 15 | 0.12 | 97 |
| | 16 | 0.112 | |
| | 17 | 0.124 | 96.5 |
| | 18 | 0.128 | |
| | 19 | 0.128 | |
| | 20 | 0.124 | |
| | 21 | 0.14 | 96 |
| | 22 | 0.156 | |
| | 23 | 0.1721 | 96.5 |
| | 24 | 0.1728 | |
| | 25 | 0.1754 | 96 |
| | 26 | 0.1767 | |
| | 27 | 0.1746 | |
| | 28 | 0.2059 | |
| | 29 | 0.1549 | |
| | 30 | 0.156 | |
| | 31 | 0.136 | |
| | 32 | 0.156 | |
| | 33 | 0.148 | |

## 5.2 Results For Iris Data-Set

Iris Tree:

```
feature = 2 threshold = -0.45762
    leaf = [1. 0. 0.]
    feature = 2 threshold = 0.30508
        feature = 3 threshold = 0.29166
            leaf = [0. 1. 0.]
            feature = 1 threshold = -0.08333
                leaf = [0. 0. 1.]
                leaf = [0. 1. 0.]
        feature = 3 threshold = 0.33333
            feature = 0 threshold = -0.02777
                leaf = [0. 1. 0.]
                leaf = [0. 0. 1.]
            leaf = [0. 0. 1.]
```

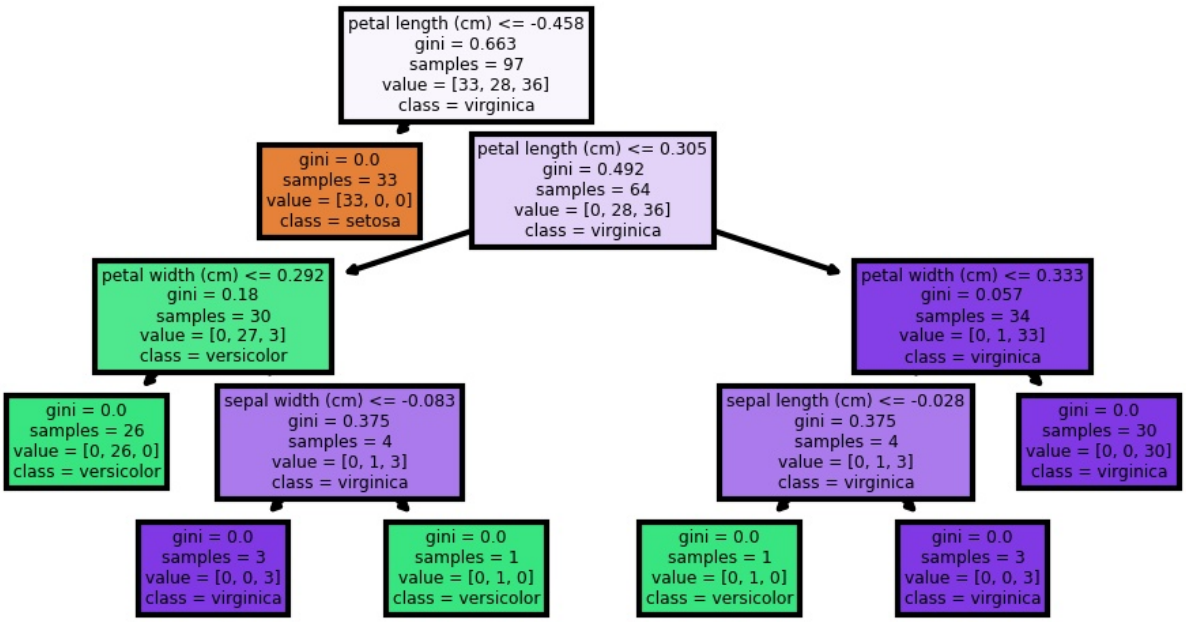


Figure 8: Iris Tree Figure (Before 1 Hot Encoding) With Max Depth = 4

Table 8:

| **Dataset: Iris** | | | |
|---|---|---|---|
| **scikit learn score = 92.45283%** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0 | 0 | 0.018 | 32.0755 |
| | 1 | 0.015 | 90.566 |
| | 2 | 0.018 | |
| | 3 | 0.019 | 92.4528 |
| | 4 | 0.016 | |
| | 5 | 0.0279 | |
| | 6 | 0.018 | |
| | 7 | 0.017 | |
| | 8 | 0.017 | |
| | 9 | 0.016 | 94.3396 |
| | 10 | 0.015 | |
| | 11 | 0.0159 | |
| | 12 | 0.015 | |
| | 13 | 0.0189 | 92.4528 |
| | 14 | 0.019 | |
| | 15 | 0.0239 | |
| | 16 | 0.0199 | |
| | 17 | 0.0249 | |
| | 18 | 0.018 | |
| | 19 | 0.0179 | |
| | 20 | 0.0209 | |
| | 21 | 0.0189 | |
| | 22 | 0.0209 | |
| | 23 | 0.0249 | |
| | 24 | 0.0309 | |
| | 25 | 0.0259 | |
| | 26 | 0.024 | |
| | 27 | 0.0219 | |
| | 28 | 0.0209 | |
| | 29 | 0.0219 | |
| | 30 | 0.02 | |
| | 31 | 0.0219 | |
| | 32 | 0.0299 | |
| | 33 | 0.0349 | |

Table 9:

| **Dataset: Iris** | | | |
|---|---|---|---|
| **scikit learn score = 92.45283%** | | | |
| **max depth = 4** | | | |
| **Window** | **Polynom Degree** | **Runtime (sec)** | **Score (%)** |
| 0.25 | 0 | 0.0439 | 32.0755 |
| | 1 | 0.0259 | 90.566 |
| | 2 | 0.017 | |
| | 3 | 0.0149 | 92.4528 |
| | 4 | 0.016 | |
| | 5 | 0.0289 | |
| | 6 | 0.0339 | |
| | 7 | 0.0229 | |
| | 8 | 0.0289 | |
| | 9 | 0.0299 | 94.3396 |
| | 10 | 0.0369 | |
| | 11 | 0.0219 | |
| | 12 | 0.0279 | |
| | 13 | 0.0439 | 92.4528 |
| | 14 | 0.0329 | |
| | 15 | 0.0489 | |
| | 16 | 0.0379 | |
| | 17 | 0.0349 | |
| | 18 | 0.0219 | |
| | 19 | 0.0189 | |
| | 20 | 0.0209 | |
| | 21 | 0.0219 | |
| | 22 | 0.0209 | |
| | 23 | 0.0489 | |
| | 24 | 0.1007 | |
| | 25 | 0.0718 | |
| | 26 | 0.0349 | |
| | 27 | 0.0349 | |
| | 28 | 0.0479 | |
| | 29 | 0.0329 | |
| | 30 | 0.0449 | |
| | 31 | 0.0349 | |
| | 32 | 0.0559 | |
| | 33 | 0.0499 | |

Table 10:

| **Dataset: Iris** | | | |
|---|---|---|---|
| **scikit learn score = 92.45283%** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.5 | 0 | 0.0429 | 32.0755 |
| | 1 | 0.022 | 90.566 |
| | 2 | 0.0229 | |
| | 3 | 0.0209 | 92.4528 |
| | 4 | 0.0199 | |
| | 5 | 0.0219 | |
| | 6 | 0.0239 | |
| | 7 | 0.0219 | |
| | 8 | 0.0329 | |
| | 9 | 0.0249 | 94.3396 |
| | 10 | 0.0249 | |
| | 11 | 0.0379 | |
| | 12 | 0.0349 | |
| | 13 | 0.0299 | 92.4528 |
| | 14 | 0.0239 | |
| | 15 | 0.0239 | |
| | 16 | 0.0319 | |
| | 17 | 0.0279 | |
| | 18 | 0.0219 | |
| | 19 | 0.0219 | |
| | 20 | 0.0199 | |
| | 21 | 0.0189 | |
| | 22 | 0.0748 | |
| | 23 | 0.1237 | |
| | 24 | 0.0219 | |
| | 25 | 0.0259 | |
| | 26 | 0.0229 | |
| | 27 | 0.0279 | |
| | 28 | 0.0249 | |
| | 29 | 0.0239 | |
| | 30 | 0.0279 | |
| | 31 | 0.0349 | |
| | 32 | 0.026 | |
| | 33 | 0.0209 | |

Table 11:

| **Dataset: Iris** | | | |
|---|---|---|---|
| **scikit learn score = 92.45283%** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.75 | 0 | 0.0259 | 32.0755 |
| | 1 | 0.0319 | 90.566 |
| | 2 | 0.0269 | |
| | 3 | 0.0279 | 92.4528 |
| | 4 | 0.0309 | |
| | 5 | 0.0299 | |
| | 6 | 0.0269 | |
| | 7 | 0.0239 | |
| | 8 | 0.0199 | |
| | 9 | 0.0319 | 94.3396 |
| | 10 | 0.0239 | |
| | 11 | 0.0199 | |
| | 12 | 0.0279 | |
| | 13 | 0.0219 | 92.4528 |
| | 14 | 0.02 | |
| | 15 | 0.0249 | |
| | 16 | 0.0229 | |
| | 17 | 0.0239 | |
| | 18 | 0.0229 | |
| | 19 | 0.0219 | |
| | 20 | 0.0239 | |
| | 21 | 0.0239 | |
| | 22 | 0.0219 | |
| | 23 | 0.0349 | |
| | 24 | 0.0369 | |
| | 25 | 0.0419 | |
| | 26 | 0.0299 | |
| | 27 | 0.0279 | |
| | 28 | 0.0289 | |
| | 29 | 0.0369 | |
| | 30 | 0.0399 | |
| | 31 | 0.0309 | |
| | 32 | 0.0269 | |
| | 33 | 0.0289 | |

### 5.3 Results For Wine Data-Set

Wine Tree:

feature = 6 threshold = -0.16877
    feature = 9 threshold = -0.56569
        leaf = [0. 1. 0.]
        feature = 6 threshold = -0.55274
            leaf = [0. 0. 1.]
            feature = 10 threshold = -0.61788
                leaf = [0. 0. 1.]
                leaf = [0. 1. 0.]
    feature = 12 threshold = -0.36305
        feature = 1 threshold = 0.23320
            leaf = [0. 1. 0.]
            leaf = [1. 0. 0.]
        feature = 4 threshold = 0.42391
            leaf = [1. 0. 0.]
            leaf = [0. 1. 0.]

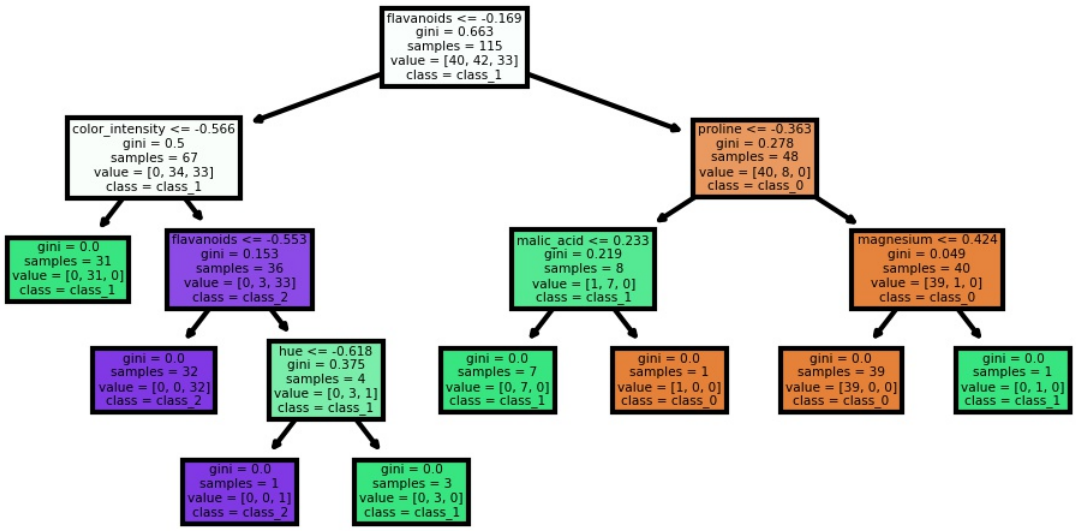


Figure 9: Wine Tree Figure (Before 1 Hot Encoding) With Max Depth = 4

Table 12:

| **Dataset: Wine** | | | |
|---|---|---|---|
| scikit learn score = 93.65079 % | | | |
| max depth = 4 | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0 | 0 | 0.0152 | 46.0317 |
| | 1 | 0.0112 | 55.5556 |
| | 2 | 0.0126 | |
| | 3 | 0.0116 | 76.1905 |
| | 4 | 0.0126 | |
| | 5 | 0.012 | 84.127 |
| | 6 | 0.0122 | |
| | 7 | 0.0124 | 92.0635 |
| | 8 | 0.0127 | |
| | 9 | 0.0247 | |
| | 10 | 0.0225 | |
| | 11 | 0.0203 | 95.2381 |
| | 12 | 0.0176 | |
| | 13 | 0.0178 | |
| | 14 | 0.0175 | |
| | 15 | 0.0157 | |
| | 16 | 0.0145 | |
| | 17 | 0.0147 | |
| | 18 | 0.0148 | |
| | 19 | 0.0152 | |
| | 20 | 0.0155 | |
| | 21 | 0.022 | |
| | 22 | 0.0219 | |
| | 23 | 0.0193 | |
| | 24 | 0.0225 | |
| | 25 | 0.0235 | |
| | 26 | 0.0178 | |
| | 27 | 0.0167 | |
| | 28 | 0.0166 | |
| | 29 | 0.0172 | |
| | 30 | 0.0173 | |
| | 31 | 0.0173 | |
| | 32 | 0.0233 | |
| | 33 | 0.0226 | |
| | 34 | 0.0234 | |

Table 13:

| **Dataset: Wine** | | | |
|---|---|---|---|
| **scikit learn score = 93.65079 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.25 | 0 | 0.0159 | 46.0317 |
| | 1 | 0.0135 | 55.5556 |
| | 2 | 0.0146 | |
| | 3 | 0.0131 | 76.1905 |
| | 4 | 0.012 | |
| | 5 | 0.0124 | 84.127 |
| | 6 | 0.0123 | |
| | 7 | 0.0126 | 92.0635 |
| | 8 | 0.0131 | |
| | 9 | 0.0131 | |
| | 10 | 0.0154 | |
| | 11 | 0.0196 | 95.2381 |
| | 12 | 0.0175 | |
| | 13 | 0.0187 | |
| | 14 | 0.0202 | |
| | 15 | 0.0187 | |
| | 16 | 0.017 | |
| | 17 | 0.0148 | |
| | 18 | 0.0149 | |
| | 19 | 0.0153 | |
| | 20 | 0.0153 | |
| | 21 | 0.0152 | |
| | 22 | 0.0221 | |
| | 23 | 0.0205 | |
| | 24 | 0.0226 | |
| | 25 | 0.0205 | |
| | 26 | 0.02 | |
| | 27 | 0.0169 | |
| | 28 | 0.0167 | |
| | 29 | 0.0173 | |
| | 30 | 0.0176 | |
| | 31 | 0.0174 | |
| | 32 | 0.0175 | |
| | 33 | 0.0254 | |
| | 34 | 0.027 | |

Table 14:

| **Dataset: Wine** | | | |
|---|---|---|---|
| **scikit learn score = 93.65079 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.5 | 0 | 0.0147 | 46.0317 |
| | 1 | 0.0259 | 55.5556 |
| | 2 | 0.025 | |
| | 3 | 0.0165 | 76.1905 |
| | 4 | 0.0123 | |
| | 5 | 0.0128 | 84.127 |
| | 6 | 0.0132 | |
| | 7 | 0.0134 | 92.0635 |
| | 8 | 0.0136 | |
| | 9 | 0.0136 | |
| | 10 | 0.0212 | |
| | 11 | 0.0159 | 95.2381 |
| | 12 | 0.0165 | |
| | 13 | 0.0182 | |
| | 14 | 0.0173 | |
| | 15 | 0.0238 | |
| | 16 | 0.0153 | |
| | 17 | 0.0152 | |
| | 18 | 0.0156 | |
| | 19 | 0.016 | |
| | 20 | 0.0162 | |
| | 21 | 0.0206 | |
| | 22 | 0.0191 | |
| | 23 | 0.0214 | |
| | 24 | 0.0243 | |
| | 25 | 0.0215 | |
| | 26 | 0.0185 | |
| | 27 | 0.0178 | |
| | 28 | 0.0174 | |
| | 29 | 0.0176 | |
| | 30 | 0.0179 | |
| | 31 | 0.0182 | |
| | 32 | 0.0249 | |
| | 33 | 0.0227 | |
| | 34 | 0.0202 | |

Table 15:

| **Dataset: Wine** | | | |
|---|---|---|---|
| **scikit learn score = 93.65079 %** | | | |
| **max depth = 4** | | | |
| Window | Polynom Degree | Runtime (sec) | Score (%) |
| 0.75 | 0 | 0.0122 | 46.0317 |
| | 1 | 0.0144 | 55.5556 |
| | 2 | 0.0149 | |
| | 3 | 0.0125 | 76.1905 |
| | 4 | 0.0119 | |
| | 5 | 0.0121 | 84.127 |
| | 6 | 0.0121 | |
| | 7 | 0.0124 | 92.0635 |
| | 8 | 0.0126 | |
| | 9 | 0.0129 | |
| | 10 | 0.013 | |
| | 11 | 0.0175 | 95.2381 |
| | 12 | 0.0185 | |
| | 13 | 0.0197 | |
| | 14 | 0.0173 | |
| | 15 | 0.0184 | |
| | 16 | 0.0211 | |
| | 17 | 0.0165 | |
| | 18 | 0.0163 | |
| | 19 | 0.0161 | |
| | 20 | 0.0159 | |
| | 21 | 0.016 | |
| | 22 | 0.0161 | |
| | 23 | 0.0198 | |
| | 24 | 0.0217 | |
| | 25 | 0.0219 | |
| | 26 | 0.023 | |
| | 27 | 0.0239 | |
| | 28 | 0.0172 | |
| | 29 | 0.0178 | |
| | 30 | 0.0173 | |
| | 31 | 0.0176 | |
| | 32 | 0.0178 | |
| | 33 | 0.0258 | |
| | 34 | 0.0243 | |

### 5.4 Conclusions

Finally we derive from the results above, the desired conclusion:
⋆The greater the tree's depth, the more branched which leads to a higher score.
⋆For all the 4 windows: [0, 0.25, 0.5, 0.75] that sent to the polynom we got the same accuracy results for all polynoms from degree 0 to degree 33, but they different in the running time.
⋆The greater the window, the higher the runtime by an approx difference of 1%.
⋆The greater the polynom's degree, the higher the score for Algorithm 1. Occasionally happens that the score is decreased by at most 3%.
⋆After several code runs, the average polynomial degree that gave us the best score is 15.
⋆According to Figure 4,5 and 6 we can observe that our score results are similar to scikit-learn results. For example, in Figure 5, when the tree's depth is in range [3,5] both results are identical.
⋆Since Cancer tree is more branched, we can observe that the accuracy score is higher than other's datasets.

## 6 Platforms

- Jupyter lab
- pycharm, python 3.9
- overleaf.com
- excel