assignment2

January 24, 2025

0.1 Introduction to Machine Learning

0.2 Assignment 2: Decision Trees

You can't learn technical subjects without hands-on practice. The assignments are an important part of the course. To submit this assignment you will need to make sure that you save your work before closing Jupyter notebook and submit your ipynb file on Blackboard

0.2.1 Assignment Learning Goals:

By the end of the assignment, students are expected to:

- Broadly describe how decision trees make predictions.
- Use `DecisionTreeClassifier()` and `DecisionTreeRegressor()` to build decision trees using scikit-learn.
- Use the `.fit()` and `.predict()` paradigm and use `.score()` method of ML models.
- Explain the concept of decision boundaries.
- Build a decision tree classifier on a real-world dataset and explore different hyperparameters of the classifier.
- Explain how decision boundaries change with `max_depth`.
- Build a decision tree regressor.

Any place you see `...`, you must fill in the function, variable, or data to complete the code. Substitute the `None` with your completed code and answers then proceed to run the cell!

Note that some of the questions in this assignment will have hidden tests. This means that no feedback will be given as to the correctness of your solution. It will be left up to you to decide if your answer is sufficiently correct. These questions are worth 2 points.

[1]:
```
# Import libraries needed for this Assignment
from hashlib import sha1

import altair as alt
import graphviz
import numpy as np
import pandas as pd
from IPython.display import HTML
from sklearn import tree
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score, cross_validate,
 ↪train_test_split
```

```python
from sklearn.tree import DecisionTreeClassifier
import test_assignment2 as t

# from display_tree import display_tree # pip install displaytree https://pypi.
↪org/project/displaytree/
alt.renderers.enable('html')
```

[1]: RendererRegistry.enable('html')

0.3 1. Decision Tree Structure

Question 1.1 {points: 5}

Label the 4 components of the decision tree diagram each with one of the possible values:

- Stump
- Root
- Branch
- Trunk
- Node
- Leaf
- Bark
- Nodule

Answer in the cell below by assigning the name of the decision tree as a string to the objects named ***label_1***, ***label_2***, ***label_3*** *and* ***label_4***.

[3]:
```python
label_1 = "Root"
label_2 = "Branch"
label_3 = "Leaf"
label_4 = "Node"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[5]:
```python
t.test_1_1_1(label_1)
```

[5]: 'Success'

[7]:
```python
t.test_1_1_2(label_2)
```

[7]: 'Success'

[9]:
```python
# check that the variable exists
assert 'label_3' in globals(
), "Please make sure that your solution is named 'label_3'"
```

```
# This test has been intentionally hidden. It will be up to you to decide if␣
↪your solution
# is sufficiently good.
```

[11]:
```
t.test_1_1_4(label_4)
```

[11]: 'Success'

Question 1.2 {points: 1}

What would this decision tree predict for an observation with the following features?

	attack	defense	sp_attack	sp_defense	speed	capture_rt	gen
0	33	101	52	23	74	12	5

Save you answer as a string in an object named ***pokemon_prediction.***

[13]:
```
pokemon_prediction = "Reg"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[15]:
```
t.test_1_2(pokemon_prediction)
```

[15]: 'Success'

Question 1.3 {points: 1}

What is the depth of the decision tree in **Question 1.2**?

Answer in the cell below with your answer and assign it to an object called ***tree_depth.***

[17]:
```
tree_depth = 4

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[19]:
```
t.test_1_3(tree_depth)
```

[19]: 'Success'

0.4 2. Decision Tree Building

Suppose you have three different job offers with comparable salaries and job descriptions. You want to decide which one to accept, and you want to make this decision based on which job is likely to make you happy. Being a very systematic person, you come up with three features associated with the offers, which are important for your happiness: whether the colleagues are supportive, work-hour flexibility, and whether the company is a start-up or not.

[54]:
```python
offer_data = {
    # Features
    "supportive_colleagues": [1, 0, 0],
    "work_hour_flexibility": [0, 0, 1],
    "start_up": [0, 1, 1],
    # Target
    "target": ["?", "?", "?"],
}

offer_df = pd.DataFrame(offer_data)
offer_df
```

[54]:

	supportive_colleagues	work_hour_flexibility	start_up	target
0	1	0	0	?
1	0	0	1	?
2	0	1	1	?

Next, you ask the following questions to some of your friends (who you think have similar notions of happiness) regarding their jobs:

1. Do you have supportive colleagues? (1 for 'yes' and 0 for 'no')
2. Do you have flexible work hours? (1 for 'yes' and 0 for 'no')
3. Do you work for a start-up? (1 for 'start up' and 0 for 'non start up')
4. Are you happy in your job? (happy or unhappy)

You get the following data from this toy survey. Your goal is to train a machine learning model using this toy data and then use this model to predict which job is likely to make you happy.

[56]:
```python
import pandas as pd

happiness_data = {
    # Features
    "supportive_colleagues": [1, 1, 1, 0, 0, 1, 1, 0, 1, 0],
    "work_hour_flexibility": [1, 1, 0, 1, 1, 0, 1, 0, 0, 0],
    "start_up": [1, 0, 1, 0, 1, 0, 0, 1, 1, 0],
    # Target
    "target": [
        "happy",
        "happy",
        "happy",
        "unhappy",
        "unhappy",
        "happy",
        "happy",
        "unhappy",
        "unhappy",
        "unhappy",
    ],
}
```

```
train_df = pd.DataFrame(happiness_data)
train_df
```

[56]:

	supportive_colleagues	work_hour_flexibility	start_up	target
0	1	1	1	happy
1	1	1	0	happy
2	1	0	1	happy
3	0	1	0	unhappy
4	0	1	1	unhappy
5	1	0	0	happy
6	1	1	0	happy
7	0	0	1	unhappy
8	1	0	1	unhappy
9	0	0	0	unhappy

Question 2.1 {points: 2}

With this toy dataset, build a decision stump (decision tree with only 1 split) by hand by splitting on the condition `supportive_colleagues` == 1.

What training accuracy would you get with this decision stump?

Save the accuracy as a fraction in an object named `supportive_colleagues_acc`.

[64]:

```
# Split the data based on supportive_colleagues
supportive_df = train_df[train_df['supportive_colleagues'] == 1]
not_supportive_df = train_df[train_df['supportive_colleagues'] == 0]

# Get the most frequent class in each split
supportive_pred = supportive_df['target'].mode()[0]
not_supportive_pred = not_supportive_df['target'].mode()[0]

# Calculate accuracy
correct_supportive = (supportive_df['target'] == supportive_pred).sum()
correct_not_supportive = (not_supportive_df['target'] == not_supportive_pred).
 ↪sum()

total_correct = correct_supportive + correct_not_supportive
total_samples = len(train_df)

supportive_colleagues_acc = total_correct / total_samples
print(f"Training Accuracy: {supportive_colleagues_acc}")

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

```
Training Accuracy: 0.9
```

[66]:
```
# check that the variable exists
assert 'supportive_colleagues_acc' in globals(
), "Please make sure that your solution is named 'supportive_colleagues_acc'"

# This test has been intentionally hidden. It will be up to you to decide if␣
↪your solution
# is sufficiently good.
```

Question 2.2 {points: 1}

The idea of a machine learning algorithm is to *fit* the best model on the given training data, which is in the form of feature vectors (`X`) and their corresponding targets(`y`), and then using this model to *predict* targets for new examples (represented with feature vectors).

From `train_df`, create the feature table and save it in an object named `X` and the target in an object named `y`.

[68]:
```
X = train_df[['supportive_colleagues', 'work_hour_flexibility', 'start_up']]
y = train_df['target']

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[70]:
```
t.test_2_2(X,y)
```

[70]: 'Success'

Question 2.3 {points: 1}

Build a decision tree named `toy_tree` and fit it on the toy data using `sklearn`'s DecisionTreeClassifier.

[78]:
```
# Create and train the decision tree
toy_tree = DecisionTreeClassifier(random_state=42)
toy_tree.fit(X, y)

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[78]: DecisionTreeClassifier(random_state=42)

[80]:
```
t.test_2_3(toy_tree)
```

[80]: 'Success'

Question 2.4 {points: 1}

Visualize the trained decision tree using the function `display_tree` that we have imported from the `display_tree` library already. Save it in an object named `toy_displayed`.

Hint: use `?display_tree` to get more information about the function.

[21]:
```python
from display_tree import display_tree

toy_displayed = None

# Visualizing the decision tree using display_tree
toy_displayed = display_tree(
    feature_names=X.columns.tolist(),  # Provide feature names as a list
    tree=toy_tree,                     # The decision tree model
    out_file="toy_tree"                # Specify the output file name (e.g.,␣
 ↪"toy_tree")
)

# To display the tree inline in a Jupyter Notebook
toy_displayed.view()

# raise NotImplementedError # No Answer - remove if you provide an answer
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[21], line 1
----> 1 from display_tree import display_tree
      3 toy_displayed = None
      5 # Visualizing the decision tree using display_tree

ModuleNotFoundError: No module named 'display_tree'
```

[23]:
```python
t.test_2_4(toy_displayed)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 t.test_2_4(toy_displayed)

NameError: name 'toy_displayed' is not defined
```

Question 2.5 {points: 1}

Score the decision tree on the training data (X and y). Save the results in an object named toy_score.

[164]:
```python
# Score the decision tree on the training data
toy_score = toy_tree.score(X, y)
print(f"Training Accuracy: {toy_score}")

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

```
Training Accuracy: 0.9
```

[166]:
```
t.test_2_5(toy_score)
```

[166]:
```
'Success'
```

Question 2.6 {points: 1}

Predict on `X`. Add the results as a column named `predicted` in the `train_df` and name this new dataframe `predicted_train`.

[174]:
```
# Make predictions on the training data
train_df['predicted'] = toy_tree.predict(X)

# Save the updated dataframe
predicted_train = train_df

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[176]:
```
t.test_2_6(predicted_train)
```

[176]:
```
'Success'
```

Question 2.7 {points: 1}

Do you get perfect training accuracy?

A) Yes, the model correctly predicts every single observation

B) No, the model made a mistake likely because the decision tree wasn't complex enough.

C) No, there are two examples in the dataset with exactly the same feature values but different targets so the model makes a mistake on one of them.

D No, the model is randomly predicting and therefore it won't get every single example correct.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer2_7`.

[180]:
```
answer2_7 = "c"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[182]:
```
t.test_2_7(answer2_7)
```

[182]:
```
'Success'
```

Question 2.8 {points: 1}

Create a feature table from the `offer_df` (We don't know the target value in this case).

Save this in an object named `offer_X`.

[184]:
```
offer_X = offer_df[['supportive_colleagues', 'work_hour_flexibility',␣
↪'start_up']]

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[186]:
```
t.test_2_8(offer_X)
```

[186]: 'Success'

Question 2.9 {points: 1}

Use the model `toy_tree` to predict which jobs from the `offer_df`, you will be happy working. In other words, `predict` on `offer_X`.

Add a column to the `offer_df` dataframe named `predicted` and save the whole dataframe in an object named `pred_offer_df`.

[194]:
```
# Make predictions
offer_df['predicted'] = toy_tree.predict(offer_X)

# Save the updated dataframe
pred_offer_df = offer_df

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[196]:
```
t.test_2_9(pred_offer_df)
```

[196]: 'Success'

1 3. Exploratory Data Analysis and Decision Trees

For the rest of the assignment you'll be using a modified version of Kaggle's Pokemon dataset. The dataset contains a number of features of pokemon's strength and weaknesses:

- `num`: ID for each Pokémon.
- `name`: Name of each Pokémon.
- `type`: Each Pokémon has a type, this determines weakness/resistance to attacks.
- `hp`: Hit points, or health, defines how much damage a Pokémon can withstand before fainting.
- `attack`: The base modifier for normal attacks (eg. Scratch, Punch).
- `defense`: The base damage resistance against normal attacks.
- `sp_atk`: Special attack, the base modifier for special attacks (e.g. fire blast, bubble beam).
- `sp_def`: The base damage resistance against special attacks.
- `total`: Sum of the `attack`, `defense`, `sp_atk`, and `sp_def` columns
- `speed`: Determines which Pokémon attacks first each round.
- `generation`: Number of generation.
- `legendary`: 1 if Legendary Pokémon, 0 if not.

In this question, our target is the `lengendary` column.

[198]:
```python
pokemon = pd.read_csv('data/pokemon.csv')
pokemon.head()
```

[198]:
```
   num                     name  hp  attack  defense  sp_atk  sp_def  speed  \
0    1                Bulbasaur  45      49       49      65      65     45
1    2                  Ivysaur  60      62       63      80      80     60
2    3                 Venusaur  80      82       83     100     100     80
3    3    VenusaurMega Venusaur  80     100      123     122     120     80
4    4               Charmander  39      52       43      60      50     65

   total  generation  legendary   type
0    228           1          0  Grass
1    285           1          0  Grass
2    365           1          0  Grass
3    465           1          0  Grass
4    205           1          0   Fire
```

Question 3.1 {points: 1}

Show information of each feature using `pd.DataFrame.info` on `pokemon` and answer the question below.

Select all that apply?

A) There are `13` columns in the dataset.

B) The legendary column is of Dtype `int64`.

C) `5` columns have null values.

D The name column is of Dtype `string`.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer(s) between `""` in a list, assign the correct answer to an object called `answer3_1`. For example ["A","B"] is a possible answer

[218]:
```python
# your code here
# Check the DataFrame information
pokemon.info()
# Check for null values in each column
pokemon.isnull().sum()
answer3_1 = ["B","D"]

# raise NotImplementedError # No Answer - remove if you provide an answer
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   num         800 non-null    int64
 1   name        800 non-null    object
```

```
 2   hp          800 non-null    int64
 3   attack      800 non-null    int64
 4   defense     800 non-null    int64
 5   sp_atk      800 non-null    int64
 6   sp_def      800 non-null    int64
 7   speed       800 non-null    int64
 8   total       800 non-null    int64
 9   generation  800 non-null    int64
 10  legendary   800 non-null    int64
 11  type        800 non-null    object
dtypes: int64(10), object(2)
memory usage: 75.1+ KB
```

[220]:
```
t.test_3_1(answer3_1)
```

[220]:
```
'Success'
```

Question 3.2 {points: 1}

Show summary statistics of each feature using `pd.DataFrame.describe` on `pokemon` and store it into a variable called `pokemon_summary`.

[224]:
```
# your code here
# Calculate summary statistics and store them in pokemon_summary
pokemon_summary = pokemon.describe()

# raise NotImplementedError # No Answer - remove if you provide an answer
```

[226]:
```
t.test_3_2(pokemon_summary)
```

[226]:
```
'Success'
```

Question 3.3 {points: 1}

Using the Altair skills, Take the code below that started for you (between the `'''`) and copy it into the solution cell. Fill in the blank areas (`....`) so that the code produces histograms for the following features (in order) that show the distribution of the feature values, separated for 0 and 1 target values.

- `hp`
- `attack`
- `defense`
- `sp_atk`
- `sp_def`
- `speed`
- `total`

```
def plot_histogram(df,feature):
    """
    plots a histogram of a decision trees feature
```

```
    Parameters
    ----------
    feature: str
        the feature name
    Returns
    -------
    altair.vegalite.v3.api.Chart
        an Altair histogram
    """
    histogram = alt.Chart(df).mark_bar(
        opacity=0.7).encode(
        alt.X(feature, bin=alt.Bin(maxbins=50)),
        alt.Y('count()', stack=None),
        alt.Color(....)).properties(
        title= str.title(feature))
    return ....

feature_list = ....
figure_dict = dict()
for feature in .... :
    figure_dict.update({feature:plot_histogram(....,feature)})
figure_panel = alt.vconcat(*figure_dict.values())
figure_panel
```

[25]:
```
def plot_histogram(df, feature):
    """
    Plots a histogram of a decision tree's feature.

    Parameters
    ----------
    df : DataFrame
        The DataFrame containing the data.
    feature : str
        The feature name.

    Returns
    -------
    altair.vegalite.v3.api.Chart
        An Altair histogram.
    """
    histogram = alt.Chart(df).mark_bar(opacity=0.7).encode(
        alt.X(feature, bin=alt.Bin(maxbins=50)),
        alt.Y('count()', stack=None),
        alt.Color('legendary:N')
    ).properties(
        title=str(feature)
    )
```

```
    return histogram

# Define the list of features to plot histograms for
feature_list = ['hp', 'attack', 'defense', 'sp_atk', 'sp_def', 'speed', 'total']

# Initialize a dictionary to store individual histograms
figure_dict = {}

# Generate and update the dictionary with histograms for each feature
for feature in feature_list:
    figure_dict.update({feature: plot_histogram(pokemon, feature)})

# Combine the histograms into a single panel for visualization
figure_panel = alt.vconcat(*figure_dict.values())

# Display the panel
figure_panel
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[25], line 34
     32 # Generate and update the dictionary with histograms for each feature
     33 for feature in feature_list:
---> 34     figure_dict.update({feature: plot_histogram(pokemon, feature)})
     36 # Combine the histograms into a single panel for visualization
     37 figure_panel = alt.vconcat(*figure_dict.values())

NameError: name 'pokemon' is not defined
```

[10]:
```
t.test_3_3(plot_histogram,figure_panel,figure_dict)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 t.test_3_3(plot_histogram,figure_panel,figure_dict)

NameError: name 't' is not defined
```

Question 3.4 {points: 2}

Which feature appears to be the most useful in differentiating the target classes?

Answer in the cell below by putting the feature name between `""` *and assign it to an object called* `answer3_4`.

[244]:
```
answer3_4 = "total"
# your code here
#raise NotImplementedError # No Answer - remove if you provide an answer
```

[246]:
```
# check that the variable exists
assert 'answer3_4' in globals(
), "Please make sure that your solution is named 'answer3_4'"

# This test has been intentionally hidden. It will be up to you to decide if␣
↪your solution
# is sufficiently good.
```

Question 3.5 {points: 1}

Suppose for a particular feature, the histograms of that feature are identical for the two target classes. Does that mean the feature is not useful for predicting the target class?

A) If the histograms are identical then there is no way differentiate each target value and so the feature is not useful.

B) If the histograms are identical then we only need to use that feature for predicting the target value.

C) If the histograms are identical, the feature might still be useful because it may be predictive in conjunction with other features.

D) If the histograms are identical, the feature might still be useful but only with other models.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`, assign the correct answer to an object called `answer3_5`.

[248]:
```
answer3_5 = "c"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[250]:
```
t.test_3_5(answer3_5)
```

[250]:
```
'Success'
```

Question 3.6 {points: 1}

Note that the dataset includes a categorical features labeled `type`. Do you think this feature could be useful in predicting whether the pokemon was legendary or not and would there be any difficulty in using it in our decision tree?

A) Yes, it would be useful but adding categorical features into a model needs special attention.

B) Yes, it would be useful and we shouldn't have any difficulty adding them into our model.

C) No, We have enough features to predict with, the added `type` column would not add anything significant.

D) No, and categorical features would need special attention to add to our model.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between "", assign the correct answer to an object called ***answer3_6****.*

[254]:
```
answer3_6 = "A"

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[256]:
```
t.test_3_6(answer3_6)
```

[256]: 'Success'

2 4. Hyperparameters

Question 4.1 {points: 1}

Create your X and y objects so that you X dataframe contains the columns: - `hp` - `attack` - `defense` - `sp_atk` - `sp_def` - `speed` - `total` - `generation`

and your y dataframe is the `legendary` column.

Save each in the respective object names ***X*** *and* ***y***

[258]:
```
# Define the feature columns
X = pokemon[['hp', 'attack', 'defense', 'sp_atk', 'sp_def', 'speed', 'total',␣
↪'generation']]

# Define the target column
y = pokemon['legendary']

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[260]:
```
t.test_4_1(X,y)
```

[260]: 'Success'

Question 4.2 {points: 3}

In this question, you'll explore the `max_depth` hyperparameter within the range 1 to 15. See the `DecisionTreeClassifier` documentation for more details.

To do so, you will need to make a for loop for each value between 1-15 that: - Creates a model named `pokemon_tree`. - Sets the `max_depth` hyperparameter to the value it's iterating on. - Sets the argument `random_state=8`. - Fits each model on X and y. - Appends the model's score to the list depth_accuracy..

[14]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```python
# Initialize the list to store accuracy scores
depth_accuracy = []

# Iterate over max_depth values from 1 to 15
for depth in range(1, 16):
    # Create the DecisionTreeClassifier with max_depth and random_state
    pokemon_tree = DecisionTreeClassifier(max_depth=depth, random_state=8)

    # Fit the model on the entire dataset
    pokemon_tree.fit(X, y)

    # Calculate the training accuracy
    accuracy = pokemon_tree.score(X, y)

    # Append the accuracy to the list
    depth_accuracy.append(accuracy)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[14], line 13
     10 pokemon_tree = DecisionTreeClassifier(max_depth=depth, random_state=8)
     12 # Fit the model on the entire dataset
---> 13 pokemon_tree.fit(X, y)
     15 # Calculate the training accuracy
     16 accuracy = pokemon_tree.score(X, y)

NameError: name 'X' is not defined
```

[16]:
```python
t.test_4_2(depth_accuracy)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 t.test_4_2(depth_accuracy)

NameError: name 't' is not defined
```

Question 4.3 {points: 3}

Make a dataframe that contains the tree depth and scores and name it `depth_scores_df`

It should look something like this:

	max_depth	accuracy
0	1	#
1	2	#

	max_depth	accuracy
2	3	#
...	...	...
14	15	#

[272]:
```
# Create the DataFrame
depth_scores_df = pd.DataFrame({
    'max_depth': list(range(1, 16)),
    'accuracy': depth_accuracy
})

# your code here
# raise NotImplementedError # No Answer - remove if you provide an answer
```

[274]:
```
t.test_4_3(depth_scores_df)
```

[274]: 'Success'

Question 4.4 {points: 1}

Using altair, make a `mark_line()` plot which displays the depth of the decision tree on the x-axis and the `depth_accuracy` on the y-axis. Make sure it has the dimensions `width=500, height=300`. Don't forget to give it a title and the plot `depth_acc_plot`

[276]:
```
# Create the plot
depth_acc_plot = alt.Chart(depth_scores_df).mark_line().encode(
    x=alt.X('max_depth:Q', title='Max Depth'),
    y=alt.Y('accuracy:Q', title='Accuracy')
).properties(
    title='Decision Tree Depth vs. Accuracy',
    width=500,
    height=300
)
```

[278]:
```
t.test_4_4(depth_acc_plot)
```

[278]: 'Success'

3 5 Decision Tree Regressor

Let's use the real estate data set that we saw in Assignment 1 and see if we can improve our R^2 from last time.

For this question we are using a dataset obtained from The UCI Machine Learning Repository that contains the market historical data of real estate valuation collected from Sindian District, New Taipei City in Taiwan.

The columns in the dataset can be explained as follows:

- `date`: the transaction date (for example, 2013.250=2013 March, 2013.500=2013 June, etc.)
- `house_age`: the house age (unit: year)
- `distance_station`: the distance to the nearest Mass Rapid Transit (MRT) station (unit: meter)
- `num_stores`: the number of convenience stores in the living circle on foot (integer)(a ***living circle*** is a residential space with similar local characteristics, and daily behaviors)
- `latitude`: the geographic coordinate, latitude. (unit: degree)
- `longitude`: the geographic coordinate, longitude. (unit: degree)
- `price`: house price per unit area (10000 New Taiwan Dollar/Ping,where Ping is a local unit of area, 1 Ping = 3.3 meter squared)

[280]:
```python
housing_df = pd.read_csv('data/real_estate.csv')
housing_df.head()
```

[280]:

	house_age	distance_station	num_stores	latitude	longitude	price
0	32.0	84.87882	10	24.98298	121.54024	37.9
1	19.5	306.59470	9	24.98034	121.53951	42.2
2	13.3	561.98450	5	24.98746	121.54391	47.3
3	13.3	561.98450	5	24.98746	121.54391	54.8
4	5.0	390.56840	5	24.97937	121.54245	43.1

Question 5.1 {points: 1}

Create your X and y objects.
For the `X` dataframe make sure that you are not including `price`. Since our `y` (target) is the `price` column.

Save each in the respective object names `X` *and* `y`

[282]:
```python
# Define the features (all columns except 'price')
X = housing_df.drop(columns=['price'])

# Define the target column ('price')
y = housing_df['price']
```

[284]:
```python
t.test_5_1(X,y)
```

[284]: 'Success'

Question 5.2 {points: 1}

Build a Decision tree Regressor named `tree_reg`. Make sure to import DecisionTreeRegressor from the sklearn.tree library. Train it on the variables `X` and `y` that we made in question 5.1. Save the score in a variable named `tree_score`.

[288]:
```python
# Initialize the Decision Tree Regressor
tree_reg = DecisionTreeRegressor(random_state=8)

# Train the regressor on the dataset
tree_reg.fit(X, y)
```

```
# Compute the R² score on the training data
tree_score = tree_reg.score(X, y)
print(f"R² Score: {tree_score}")
```

```
R² Score: 0.9893300488110535
```

[290]:
```
t.test_5_2(tree_score)
```

[290]: 'Success'

Question 5.3 {points: 2}

Does the model do better than the Dummy Regressor we used in assignment 1?

A) Both models Dummy Regressor and Decision Tree Regressor do about the same.

B) Dummy Regressor does moderately better.

C) Decision Tree Regressor does moderately better.

D) Dummy Regressor does much better than the Decision Tree Regressor.

E) Decision Tree Regressor does much better than the Dummy Regressor.

Answer in the cell below using the uppercase letter associated with your answer. Place your answer between `""`*, assign the correct answer to an object called* `answer5_3`*.*

[294]:
```
answer5_3 = "E"
```

[296]:
```
# check that the variable exists
assert 'answer5_3' in globals(
), "Please make sure that your solution is named 'answer5_3'"

# This test has been intentionally hidden. It will be up to you to decide if␣
↪your solution
# is sufficiently good.
```

3.1 Before Submitting

Before submitting your assignment please do the following:

- Read through your solutions
- **Go to the file and download ipynb, file –> save and export notebook as pdf, submit both ipynb and pdf file, if you not able to perform conversion sub**
- Makes sure that none of your code is broken
- Verify that the tests from the questions you answered have obtained the output "Success"

This is a simple way to make sure that you are submitting all the variables needed to mark the assignment. This method should help avoid losing marks due to changes in your environment.

3.2 Attributions

- Fertitily Diagnosis Dataset: - The UCI Machine Learning Repository

David Gil, Jose Luis Girela, Joaquin De Juan, M. Jose Gomez-Torres, and Magnus Johnsson. Predicting seminal quality with artificial intelligence methods. Expert Systems with Applications, 39(16):12564 â€“ 12573, 2012

- Real Estate Dataset - The UCI Machine Learning Repository

```
[ ]:
```