Alessandro Allegranzi

May 15, 2024

CS 767 Assignment 1

Decision trees

[ASSIGNMENT 1: DECISION TREES 2](#_Toc166406897)

[1. SUMMARY DESCRIPTION 2](#_Toc166406899)

[2. DATA SOURCE 2](#_Toc166406901)

[3. I/O CASES 3](#_Toc166406902)

[4. KEY SOURCE CODE 4](#_Toc166406904)

[5. YOUR SOURCE 5](#_Toc166406906)

[6. DATA ALTERATION FOR CHANGED RESULTS 5](#_Toc166406908)

[7. INCONSISTENT DATA 6](#_Toc166406913)

[8. BENEFITS 7](#_Toc166406918)

[9. EVALUATION 8](#_Toc166406920)

[References 8](#_Toc166406921)

[APPENDIX 1 (if required) 8](#_Toc166406922)

[APPENDIX 2 (if required) 10](#_Toc166406923)

05/15/2024

*Alessandro Allegranzi*

# ASSIGNMENT 1: DECISION TREES

The purpose of this assignment is to reinforce the idea of decision trees hands-on.

## Please use this Word file template for your response. Follow—and retain—these instructions in gray text. Insert your work in black where indicated. Keep in mind the evaluation matrix below as you do the work and use it to guide what you submit. Complete this assignment with the assistance of an AI generator such as ChatGPT. Your work will be assessed in terms of *your value added* as per the evaluation matrix at the end. You will describe your value added in the format provided by this Word template—it consists of your prompts together with you edits and additions to AI-generated material.

You may build on the application you select (see Section 1) for subsequent assignments if you wish.

Use no more than 6 pages of 12-point text excluding figures, the instructions in gray, your AI generator descriptions, and appendices. You can add as many appendices as you like, as per the instructions provided in the first appendix below.

You may build on the work of others but (1) show clearly that you understand this work and (2) observe all plagiarism rules scrupulously, including clear citations. Use the Reference section at the end.

## 1. SUMMARY DESCRIPTION

In one or two sentences, describe a decision tree application not in the literature, that you will implement. Select an application of personal or professional interest to you. You can begin with existing code (an example is [here](https://colab.research.google.com/drive/1DGZdokU6DOXCtDjYjGU_yev0_6KZOP2o?usp=sharing), which you can copy to your Google drive).

I will implement a decision tree application with regression models that predict the temperature, likelihood of rainfall, and expected snowfall in the city of Boston on a certain day based on historical weather data from the past 24 years. Weather forecasts are extremely useful, and a decision tree trained on historical data should be accurate enough for people to plan their day. A regression model will be used since the desired output is a numerical prediction and not a classification [1].

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

## 2. DATA SOURCE

Identify and explain the source of your data (e.g., collected by hand; Kaggle). Point to a URL so we can see the data.

I requested data from the National Centers for Environmental Information. (https://www.ncdc.noaa.gov/cdo-web/search). NOAA collects an extensive amount of information from weather stations around the US and is a reliable source of publicly available weather data. It’s not possible to actually visualize the data online, but requesting a download is easy. I will attach a CSV file with my submission to show the data gathered. I selected daily temperature and precipitation data from the Boston Logan weather station from January 1, 2000 to May 9, 2024.   
  
Sample:

A table with numbers and letters

Description automatically generated with medium confidence

## 3. I/O CASES

For three different inputs (e.g., a list or array) for your application, show the outputs.

**Input 1:** panda dataframe:day of the year (formatted as 1 out of 365).

**Output 1:** Expected average temp for the day

**Example**

Input: 182

Output: 71.57708957708958

-------------------- -

**Input 2:** panda dataframe: day of the year and temperature

**Output 2:** Expected precipitation for the day.

**Example**

Input:

DayOfYear TAVG

0 105 60

Output:

0.12382053194133685, i.e - 12% chance of rain

-------------------- -

**Input 3: p**anda dataframe: day of the year and temperature

**Output 3:** Expected snowfall for the day.

**Example**

Input:

DayOfYear TAVG

0 12 28

1 213 80

Output:

[0.55285714 0.00849749], i.e: .55 inches of snowfall predicted for Jan 12 and 0 for August 1

-------------------- -

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

## 

## 4. KEY SOURCE CODE

Please supply the most relevant decision tree code).

Code for first regression tree to predict likely average temperature given day of the year:

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

# Data sourced from NOAA: https://www.ncei.noaa.gov/cdo-web/

input\_file = "/content/drive/MyDrive/Colab Notebooks/data/boston\_weather\_data.csv"

weather\_data = pd.read\_csv(input\_file)

# Uncomment to see info on data frame

# weather\_data.info()

# Training on a date is difficult because models work best with numerical inputs and

# working with datetime is not straightforward. I decided to convert datetime into

# the day of the year.

weather\_data['DATE'] = pd.to\_datetime(weather\_data['DATE'])

weather\_data['DayOfYear'] = weather\_data['DATE'].dt.dayofyear

# I got nan errors for values in y, so cleaning the data.

weather\_data = weather\_data.dropna()

# Separate features and target

features = ['DayOfYear']

X = weather\_data[features]

y = weather\_data['TAVG']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model

tree\_reg = DecisionTreeRegressor(max\_depth=3)

tree\_reg.fit(X\_train, y\_train)

# Test the model

predictions = tree\_reg.predict(X\_test)

[For other trees used please see appendix.](#appendix_one)

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

Your response replaces this.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

## 5. YOUR SOURCE

Please supply the URL of your code (e.g., as shared Colab code).

[Colab Notebook Link](https://colab.research.google.com/drive/13l4ouydDpBIzwLeN-oZWV7shWWL8OeJj?usp=sharing)

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

Your response replaces this.

## 

## 6. DATA ALTERATION FOR CHANGED RESULTS

Alter the data from your data source so that it changes in the outputs for the above inputs—but not as in Section 7 below. Explain in the format below why the output changes the way it does. Limit (for this Section 6): 2 normal paragraphs (remember that you can use appendices for reference material).

## 6.1 What Data Was Altered?

I altered the precipitation data by increasing the chance of rain for every entry.

Code snippet below:

# Copying original data frame and adding 2 to every entry for the

# precipitation column.

altered\_weather\_data = weather\_data.copy()

altered\_weather\_data['PRCP'] += .1

## 6.2 How Did the Output Change?

## 

The output has basically shifted by .1, predicting higher rainfall percentages that before the change. As an example, predicted rainfall for April 15 was 0.12382053194133685 in the original model. In the new model trained on altered data, the prediction was 0.22382053194134666, which is close to .1 higher, exactly.

## 6.3 Why Did the Output Change in This Way?

By increasing all precipitation likelihoods by 0.1, we're effectively shifting the entire precipitation likelihood distribution to the right. This means that for a given input, the corresponding precipitation likelihood is now 0.1 higher than it was before.

Since the model has learned that higher precipitation likelihoods correspond to certain outputs (depending on the specific relationship), it will now predict a different output for the same input. This is a direct result of the model's learned relationship between precipitation likelihood and output. Decision tress are known to have high variance, and even small changes in data can lead to different results [1].

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

## 

## 7. INCONSISTENT DATA

Alter the data so that it contains *inconsistencies*. Using the format below, show changes in the I/O of the examples in part 3, and explain. Limit: 3 normal paragraphs.

## 7.1 What Data Was Altered to be Inconsistent?

I altered the average temperature entries in the dataframe to use inconsistent units of measurement to see what the effect would be on the initial temperature prediction tree. I created a method to randomly alter some ‘TAVG’ cells from Fahrenheit to Celsius and ran it on the weather data dataframe.

Code snippet:

def convert\_random\_entries\_to\_celsius(temp\_f):

if np.random.rand() < 0.3: # 30% chance to convert

return (temp\_f - 32) \* 5.0/9.0

else:

return temp\_f

incosistent\_weather\_data = weather\_data.copy()

incosistent\_weather\_data['TAVG'] = incosistent\_weather\_data['TAVG'].apply(convert\_random\_entries\_to\_celsius)

## 7.2 How Did the Output Change?

## 

The output changed and showed consistently lower temperature predictions that before. As an example, the July 1 temperature prediction dropped almost 20 degrees.

## 7.3 Why Did the Output Change in This Way?

Due to conversion, Celsius values are lower than their Fahrenheit counterparts. 30% of the data being in Celsius brought prediction values down for the model across the board since the model was not able to reconcile the differences in units of measurement.

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

## 

## 8. BENEFITS

In at most 3/4 page (of 12-point text), explain the pros and cons of decision trees *applied to the application you have chosen* (i.e., don’t respond generically).

**Pro’s of decision tree usage for my application**

A Decision Tree Regressor is a powerful machine learning model that can handle both numerical and categorical data, making it a versatile choice for data sets like weather data. One of the main advantages of using a Decision Tree for weather prediction is its interpretability. Decision Trees are easy to understand and visualize, as they mimic human decision-making more closely than many other types of models. This makes it easier to explain the model's predictions and understand how different features in the data, such as the day of the year, influence the predicted weather [1].

Another advantage of using a Decision Tree Regressor is its ability to handle non-linear relationships between features and the target variable. This is particularly useful in weather prediction, where relationships between variables can be complex and non-linear. For example, the relationship between the day of the year and temperature is not linear, as temperatures tend to rise in the middle of the year and fall at the beginning and end. A Decision Tree can capture this relationship effectively. Furthermore, Decision Trees do not require extensive data preprocessing, such as normalization or scaling, and can handle missing values, which are common in weather data [1].

**Con’s of decision tree usage for my application**

While Decision Tree Regressors offer several advantages, they also have some limitations. One of the main drawbacks is their tendency to overfit, especially when dealing with complex datasets like weather data. Overfitting occurs when the model learns the training data too well, including its noise and outliers, which can lead to poor performance on unseen data. This is because the model becomes too complex, with many branches leading to decisions based on specific instances in the training data, rather than general patterns. Although techniques like pruning, setting a minimum number of samples per leaf, or limiting the maximum depth of the tree can help mitigate overfitting, it remains a challenge. The weather application trees included in this assignment have trained trees with a max depth of 3 to mitigate the danger of overfitting.

Another limitation of Decision Trees is their sensitivity to small changes in the data [1]. Sections 6 and 7 showcased how a slight alteration in the training data can result in a significantly different tree. This lack of robustness means that if the model is trained on a slightly different dataset, it might produce very different predictions. This can be problematic for weather prediction, where data can vary greatly from year to year. Additionally, Decision Trees can create biased trees if some classes dominate, and they are not ideal for estimation of tasks where the target variable takes continuous values, as trees are better suited for categorical variables.

### >>AI generation (or check: I did not use AI generation here \_X\_). Please collapse this section before submitting.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

## 9. EVALUATION



# References

If you use anyone’s work, the extent of that usage must be made very clear. Where this use is via an AI generator, please use the sections provided. Every other reference must be cited at least once within the paper above.

[1] Geron, Aurelien. *Hands-On Machine Learning with Scikit-learn, Keras, & Tensorflow Third Edition*. “Chapter 6: Decision Trees”. O’Reilly Media, Inc. October, 2022

# APPENDIX 1 (if required)

Code for second decision tree to predict rainfall:

# Separate features and target

features = ['DayOfYear', 'TAVG']

X = weather\_data[features]

y = weather\_data['PRCP']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model

rain\_tree\_reg = DecisionTreeRegressor(max\_depth=3)

rain\_tree\_reg.fit(X\_train, y\_train)

# Test the model

predictions = rain\_tree\_reg.predict(X\_test)

# Uncomment to examine predicitons

# predictions

# Convert date to day of the year so our model can work with

date = datetime.strptime('2025-04-15', '%Y-%m-%d')

day\_of\_year = date.timetuple().tm\_yday

avg\_temp = 60

# Create a DataFrame for the prediction. Consists of day of the year and average

# temp.

X\_pred = pd.DataFrame([[day\_of\_year, avg\_temp]], columns=features)

# Predict temperature

rainfall = ain\_tree\_reg.predict(X\_pred)

print(f'The predicted rainfall for April 15 is {rainfall[0]}')

Sample output: The predicted rainfall for April 15 is 0.62

Code for second decision tree to predict snowfall:

# Separate features and target

features = ['DayOfYear', 'TAVG']

X = weather\_data[features]

y = weather\_data['SNOW']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model

snow\_tree\_reg = DecisionTreeRegressor(max\_depth=3)

snow\_tree\_reg.fit(X\_train, y\_train)

# Test the model

predictions = snow\_tree\_reg.predict(X\_test)

# Uncomment to examine predictions

# predictions

# Convert date to day of the year so our model can work with

date\_one = datetime.strptime('2025-01-12', '%Y-%m-%d')

day\_of\_year\_one = date.timetuple().tm\_yday

date\_two = datetime.strptime('2025-08-01', '%Y-%m-%d')

day\_of\_year\_two = date.timetuple().tm\_yday

avg\_temp\_one = 28

avg\_temp\_two = 80

# Create a DataFrame for the prediction. Consists of day of the year and average

# temp.

X\_pred = pd.DataFrame([[day\_of\_year\_one, avg\_temp\_one],[day\_of\_year\_two, avg\_temp\_two]], columns=features)

# Predict temperature

snowfall = snow\_tree\_reg.predict(X\_pred)

print(f'The predicted snowfall in inches for January 12 and August 1 ar: {snowfall}')

sample output: The predicted rainfall for April 15 is [4.7 0. ]

# APPENDIX 2 (if required)

…